AFRL-IF-RS-TR-2005-257
Final Technical Report
July 2005

HIGH PRODUCTIVITY COMPUTING SYSTEMS ANALYSIS AND PERFORMANCE

University of Southern California at Marina del Rey

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. N856

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2005-257 has been reviewed and is approved for publication

APPROVED: /s/

CHRISTOPHER FLYNN
Project Engineer

FOR THE DIRECTOR: /s/

JAMES A. COLLINS, Acting Chief
Advanced Computing Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JULY 2005	Final Jun 02 – May 04

4. TITLE AND SUBTITLE
HIGH PRODUCTIVITY COMPUTING SYSTEMS ANALYSIS AND PERFORMANCE

5. FUNDING NUMBERS
C - F30602-02-1-0181
PE - 62301E
PR - N856
TA - HP
WU - CS

6. AUTHOR(S)
Robert F. Lucas

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Southern California/ISI
4676 Admiralty Way
Suite 1001
Marina del Rey California 90292-6695

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFTC
26 Electronic Parkway
Rome New York 13441-4514

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2005-257

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: Christopher Flynn/IFTC/(315) 330-3249/ Christopher.Flynn@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*
Due to its focus on performance, the Defense High Performance Computing (HPC) community has always had a need for performance measurement, evaluation, and comparison. Nevertheless, there have been no widely accepted standards for measuring the performance of HPC systems, much less their productivity when applied to the unique computational challenges facing Defense scientists and engineers as well as operational users.
In coordination with MITRE and Lincoln Laboratory, the University of Southern California's Information Sciences Institute (ISI) led an effort to begin addressing this issue in support of DARPA's High Productivity Computing Systems (HPCS) program. This final report describes how ISI formed a team of experts that helped enable the HPCS program to determine Defense high performance computing needs. It then describes how ISI and its subcontractors, in a supplementary effort, initiated the HPCS phase two development time and execution time productivity activities. The outcome of this project was a new understanding of the various components of the productivity of Defense. HPC systems together with a methodology for measuring productivity from both the prspective of application developers as well as users of HPC systems. In addition, a new set of HPCS benchmarks were developed, including a set of discrete mathematics kernels.

14. SUBJECT TERMS
High Performance Computing, High Productivity Computing, Metrics, Benchmarks

15. NUMBER OF PAGES
59

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

List of Tables

1. Executive Summary

The Defense High Performance Computing (HPC) community has long had a need for productivity metrics and methodologies for its computing systems. Unfortunately, there are no widely accepted standards for measuring even the performance of HPC systems (other then the discredited Linpack benchmark[1]) much less their productivity when applied to the unique computational challenges facing Defense scientists and engineers as well as operational users. In coordination with MITRE and Lincoln Laboratory, the University of Southern California's Information Sciences Institute (ISI) undertook an effort to address the lack of HPC productivity metrics and methodology as part of the Defense Advanced Research Projects Agency's (DARPA) High Productivity Computing Systems (HPCS) program.

ISI's initial role in the HPCS program was to develop, in close coordination with DARPA and the broader Defense computing community, an understanding of the needs of the Defense computing community. There are differing requirements for application developers, users of these applications, and the broad communities that must share large-scale assets such as those provided by the High Performance Computing Modernization Program (HPCMP). ISI accomplished this goal by studying the requirements of a number of Defense users, developing the HPCS benchmarks with MITRE, and representing these Defense needs to HPCS vendors and the broader community at HPCS principle investigator meetings.

The second of ISI's initial tasks was to organize a team that could define productivity and develop a consensus for it within the broader HPC community. The team included leading experts in the field from universities and the Defense industry. We leveraged existing activities already supported by the Department of Energy (DOE), the National Science Foundation (NSF), and of course, the Department of Defense (DOD). Along with the broader HPCS community, the team helped to determine what aspects of productivity were neither well defined nor measured, and proposed steps to rectify this. These results were briefed to DARPA and at HPCS Principal Investigator (PI) meetings thereby helping define the goals of the DARPA HPCS program's second phase.

At the end of its first year, the DARPA HPCS program both down-selected to three prime contractors and refocused its analysis and performance assessment team. Looking towards the second phase of HPCS, there were two major concerns. The first was to quantify the relative difficulty of developing Defense HPC software using different programming models and tools (i.e., development time). The second was to quantify the performance delivered by large-scale systems to specific Defense applications (i.e., execution time).

In order to support DARPA during the transition to the second phase of the HPCS program, the team was tasked to design for DARPA specific research projects whose goal would be to address DARPA's needs regarding metrics and methodologies for quantifying development time and execution time productivity. The team reacted to this charge by developing the HPCS execution time strategy in coordination with

researchers at Lawrence Livermore National Laboratory (LLNL) and the University of California at San Diego (UCSD). Researchers were engaged at the University of Maryland (UMD) and other leading academic institutions to not only develop a development time strategy, but to even conduct pioneering experiments to measure the productivity of software developers when confronted with various parallel programming methodologies.

The outcome of this project was a new understanding of the various components of the productivity of Defense HPC systems together with a methodology for measuring productivity from both the perspective of application developers as well as users of HPC systems. Along the way, a new set of HPCS benchmarks was developed, including a set of discrete mathematics kernels (see Table 2) released by the team. Finally, by engaging many of the leaders of the US HPC community, the team helped DARPA facilitate a broad consensus, maximizing the impact of its HPCS program.

2. Introduction

The Defense High Performance Computing community has long had a need for productivity metrics and methodologies for its computing systems. Unfortunately, there are no widely accepted standards for measuring even the performance of HPC systems (other then the discredited Linpack benchmark) much less their productivity when applied to the unique computational challenges facing Defense scientists and engineers as well as operational users. In coordination with MITRE and Lincoln Laboratory, the University of Southern California's Information Sciences Institute undertook an effort to address the lack of HPC productivity metrics and methodology as part of DARPA's HPCS program.

ISI's initial role in the HPCS program was to develop, in close coordination with DARPA and the broader Defense computing community, an understanding of what the needs of the Defense computing community are and to establish consensus within the HPCS community as to what productivity is. The result of the first phase of the HPCS Analysis project were briefed to DARPA and at HPCS PI meetings and helped define the goals of DARPA's HPCS phase two productivity team.

At the end of its first year, the DARPA HPCS program down-selected to three prime contractors and refocused its analysis and performance assessment team. Looking towards the second phase of HPCS, there were two major concerns. The first was to quantify the relative difficulty of developing Defense HPC software using different programming models and tools (i.e., development time). The second was to quantify the performance delivered by large-scale systems to specific Defense applications (i.e., execution time). Our team responded by initiating the HPCS phase two development time and execution time productivity projects.

The remainder of this final report describes how our team accomplished their goals, and is organized as follows. ISI's role the first phase of HPCS was primarily one of coordination and leadership. Thus this report is more of a history than a recitation of technical results. The initial project is discussed in Section 3, with the formation of the HPCS Analysis team presented in Section 3.1 and with the development of HPCS metrics and methodology described in Section 3.2. The supplementary effort is discussed in Section 4, with benchmarking addressed in Section 4.1, the HPCS web site in Section 4.2, development time activities presented in Section 4.3 and execution time planning described in Section 4.4. The HPCS project is ongoing, and our team's activities in support of phase two are briefly outlined in Section 5. A list of publications is contained in Section 6. The personnel involved in the HPCS Analysis project are named in Section 7. This is followed by results, conclusions, and technology transfer in Section 8. Inventions and patent disclosures, or rather the lack thereof, is discussed in Section 9. References are provided in Section 10. Section 11 contains a list of acronyms, and finally, Section 12 contains the appendices.

3. Initial HPCS Analysis Effort

The initial objective of the HPCS Analysis project was to determine the needs of the Defense HPC users, the state-of-the-art of the field of HPC benchmarking and establish a better consensus in the HPC community as to how productivity and value should be defined. This was to be accomplished by organizing a team of experts from both academe and industry. The next two sections describe how the HPCS Analysis team was organized and what it accomplished. The authors used presentations to the government, workshops, conference presentations, and other means to disseminate the results to the Defense HPC community.

3.1 Forming the HPCS Analysis Team

ISI was initially asked to participate in the HPCS program to complement an effort that Dr. Richard Games of MITRE was heading to represent the needs of the Defense HPC community to the original five HPCS system vendors. MITRE has its own expertise in areas such as reconnaissance. However, because of both the diversity of its mission as well as security restrictions, no one person or organization can be familiar with the full range of Defense needs. Therefore, ISI was added to the initial HPCS productivity team both to bring its own expertise to bear as well as to reach out to others in industry and academe where necessary. The subcontractors were chosen so as to complement the skills of ISI and MITRE as well as to give voice to those parts of the Defense community that under normal circumstances prefer to maintain their anonymity.

Many components of the Defense HPC community, and its colleagues in other government agencies, operate with full public scrutiny, and are able to voice their needs openly. For example, the Defense High Performance Computing Modernization Program (HPCMP), while it does support classified computing, is primarily an unclassified activity providing HPC infrastructure to support open research performed in academe and at Defense laboratories. Therefore, HPCMP could forthrightly represent its needs and those of its user's to the HPCS program and its vendors. The DOE's Office of Science performs only unclassified research, so it too could represent its own needs in public forums. Both HPCMP and DOE SC were (and continue to be) overt and active HPCS "mission partners".

There are also HPCS mission partners whose work is classified, and who are thus unable to directly represent their computational requirements. Our team took this into account and included investigators with appropriate access to visit these sensitive organizations. These included Dr. Barbara Yoon as well as SGI and the Northrup-Grumman Corporation (NGC). We were thus able to include in our findings the requirements of the classified Defense community as well as the open, scientific and engineering community.

The user communities discussed above tend to develop their own codes. However, most HPC users, whether in the Defense community, or outside of it, use codes developed by others. Most such codes come from government labs or independent

software vendors (ISVs), and no study of Defense HPC requirements would be complete without them. Furthermore, there has been an absence of a measure of the throughput of critical full-scale applications in the last decade. Therefore, ISI contracted with Professor David Benson of the University of California at San Diego (UCSD) to develop a benchmark that measures the performance delivered to Defense users by real a code. UCSD focused its efforts on the LS-DYNA[2] code, which is the principle computational bottleneck in the automotive mechanical computer-aided engineering (MCAE) field. This code originated at LLNL and has many properties similar to those of full-scale Defense applications.

In the first quarter of the HPCS Analysis project, much of ISI's activity was directed toward organizing the above team. Once the core team had been identified and brought under contract, the real work of determining metrics and methodologies for defining the productivity of HPCS systems could begin.

3.2 Developing HPCS Productivity Metrics and Methodology

An early goal of the HPCS program was to ascertain Defense HPC requirements so as to be able to represent them to the HPCS vendors, none of whom support the full range of Defense users. Visits were made to NSA and other intelligence agencies, DOE, and LLNL. The latter were to discuss the needs of the nuclear weapons community. Topics of discussion included both computing requirements as well as software development practices.

Initial results regarding software development practices and methodologies were presented at the HPCS PI meeting in Phoenix, AZ, in the last week of October, 2002. The PI reported on software development practices at the NSA and DOE National Nuclear Security Administration (NNSA) Labs. Commercial MCAE ISVs were included for contrast. The center-piece of this report is the data shown in Table 1, which depicts the six main nuclear weapon performance codes initiated by the NNSA, along with an older, legacy code. The size of the codes, the average number of programmers employed, the length of time the codes had been under development, and a measure of their success are all provided. An important point is that NNSA multi-physics codes contain hundreds of thousands of lines of code, are built by a dozen or more scientists and engineers, and evolve over a period of many years. Note, in spite of this investment, they are not all successful. Once these codes reach a level of maturity, they are often run for months on end, and the biggest constraint on NNSA's scientific productivity tends to be their execution time. Because such codes are often used for decades, the developers have to use very conservative languages and programming models. Often they are limited to Fortran, C, and MPI.

	LLNL			LANL			
Code Projects	ASCI A	ASCI B	Legacy A	Antero Code Project	Shavano Code Project	Crestone Code Project	Blanca Code Project
Single Lines of Code	184000	640000	410550	300000	500000	314000	200000
Function Points (Eq. 1)	4800	6100	5400	2900	4800	2900	3774
Estimated schedule(Eq.4)	8.7	9.0	6.9	6.6	8.1	6.7	7.4
Project age (at initial milestone due date)	3	9	N/A	4	3.5	8	8
Successful in achieving initial ASCI Milestone	No	Yes	N/A	No	No	Yes	No
Estimated staff requirements (Eq.3)	22	27	24	14	22	14	18
real team size (1997-2002)	20	22	8	17	8	12	35

Table 1: Software Resource Estimates for the LLNL and LANL ASCI projects (data from Dr. Douglass Post, Los Alamos National Laboratory)

NSA has application code teams of comparable size in terms of source code, people, and duration. However, it also has a very different application development paradigm. Often one person, working under tremendous time pressure, will develop a body of code to perform what is best referred to as a mathematical experiment. Many times the code is used only once, leading such programs to be called "Kleenex codes". In this latter, "lone researcher" mode of software development, the time to develop the code can be the biggest constraint on overall productivity. The need to maximize programmer productivity, together with the short life span of the Kleenex codes, allows NSA developers to be early adopters of languages such as Unified Parallel C (UPC).

While MCAE companies often employ hundreds of people, the core development teams tend to on the order of one to two dozen. Often, as with the NNSA and NSA codes, there are one or two "heroes" who propel the overall team. Like the NNSA code developers, MCAE programmers have to use conservative programming models that run on a wide variety of platforms and can be expected to be supported many years in the future. Thus, they tend to use Fortran and the Message Passing Interface (MPI). Some of the newer codes are written with C++.

Another result of the early HPCS work was the development of a set of benchmarks representing the Defense HPC computing workload with which the program could both motivate its vendors and measure their progress. ISI collaborated with MITRE's Dr. David Koester on this effort. Together, ISI and MITRE developed a strategy of mixing traditional computational-kernel benchmarks with full applications. The kernel benchmarks were to be a spanning set of small codes that measured specific aspects of

a computing system's performance. There would also be a modest set of full applications enabling vendors to study complete codes, including input and output characteristics. The HPCS benchmarking strategy evolved over the course of the project. Therefore, discussion of a later strategy is included in Section 4.1.

Table 2 contains the benchmarks that were chosen for HPCS in phase one. ISI and MITRE chose the kernel benchmarks to represent computations that are critical to Defense, and often unique to it. One and two-dimensional Fast Fourier Transforms (FFT) were chosen to represent reconnaissance. Linear solvers were chosen to represent science and engineering. A set of discrete mathematics codes used in Defense procurements was chosen to represent a large part of the classified Defense workload. Finally, a set of novel, graph analysis benchmarks was proposed to represent the increasingly important Defense data analysis problem.

The first category of HPCS kernel benchmarks was one and two-dimensional FFTs. The FFTW benchmark from MIT was chosen as it is widely known to the embedded Defense HPC community. A two-dimensional FFT code written by MITRE's Brian Sroka was also included, to represent higher dimensional FFTs that also occur in the workloads of the mission partners.

The second category of HPCS kernel benchmarks was linear solvers. The obvious choice for a dense matrix solver is High Performance Linpack (HPL), which is widely used throughout the HPCS community. As a benchmark for sparse solvers, the well known conjugate gradient code from the National Aeronautics and Space Administration's (NASA) Advanced Supercomputing Division's (NAS) Parallel Benchmarks (NPB) was chosen.

Benchmarks

Application Area	Benchmark Type	Name	Source	Additional Information
Signals	1D FFT	FFTW	Available on Web	
Remote Sensing	2D FFT	RT_2DFFT	Available from MITRE	Brian Sroka
Stockpile Stewardship	Radiation Transport	UMT2000	ASCI Purple	LLNL Radiation Transport
	Unstructured Grids			
	Eulerian Hydrocode	SAGE3D	ASCI Purple	LANL Eulerian Physics.
	Adaptive Mesh			SAIC IP
	Finite Difference Model	CTH	DoD HPCMP TI-03	SNL Engineering Physics
				Export Controlled
Ocean Forecasting	Finite Difference Model	NLOM	DoD HPCMP TI-03	
Army Future Combat Weapons Systems	Finite Difference Model	CTH	DoD HPCMP TI-03	Export Controlled
Biological	TBD	TBD	TBD	
Crashworthiness	Multi-physics Nonlinear Finite Element	LS-DYNA	Available to Vendors	Commercially Available
Linear Algebra	Lower / Upper Triangular Matrix Decomposition	LINPACK	Available on Web	Time to solution = 72 hours
	Conjugate Gradient	NAS CG C	DoD HPCMP TI-03	Solve Laplace's equation on a cubic grid
Discrete Math	Global Updates per second (GUP/S)	RandomAccess	Paper & Pencil	Contact Bob Lucas (ISI)
	Multiple Precision	none	Paper & Pencil	Contact Bob Lucas (ISI)
	Dynamic Programming	none	Paper & Pencil	Contact Bob Lucas (ISI)
	Matrix Transpose [Binary manipulation]	none	Paper & Pencil	Contact Bob Lucas (ISI)
	Integer Sort [With large multiword key]	none	Paper & Pencil	Contact Bob Lucas (ISI)
	Binary Equation Solution	none	Paper & Pencil	Contact Bob Lucas (ISI)
Graphs	Graph Extraction (Breadth First) Search	none	Paper & Pencil	
	Sort a large set	none	Paper & Pencil	
	Construct a Relationship Graph Based on Proximity	none	Paper & Pencil	

Table 2: HPCS Phase One Benchmarks

The discrete mathematics benchmarks are codes used by the intelligence community, which were released to the HPCS program after DARPA and its contractors agreed to obscure their source. They were released to the HPCS community as a set of six discrete mathematics benchmarks. These codes stress a system's ability to randomly access large memories, and to compute on unusual operands ranging from single-bit integers to very large, multi-precise integers. The most widely known of these codes is RandomAccess, which randomly updates a very large array in memory, exposing the main memory latency of modern microprocessors. The second benchmark measures performance when multiplying multi-precise integer numbers. The third is a dynamic programming kernel, which incorporates a comparison inside of a sparse matrix multiply kernel. The fourth transposes the bits in a GF(2) matrix, and is designed to reward vendors who have bit-matrix-multiply (BMM) function units. The fifth benchmark is an integer sort. The final benchmark is the solution by Gaussian elimination of a dense linear system over GF(2), again rewarding vendors who have BMM instructions.

The final category of HPCS kernel benchmarks was graph analysis. These kernels were designed to represent a class of problems of increasing importance to the intelligence community. Three kernels were proposed: graph construction; sorting of large sets; and clustering. At the end of the first phase of the HPCS Analysis project, these only existed as pencil and paper specifications.

The choice of full application codes with which to represent the Defense workload was also done as a collaboration with MITRE. MITRE surveyed the unclassified mission partners and suggested the Navy's NLOM ocean-modeling code together with something to represent Biology. Meanwhile we worked with the NNSA labs. NNSA politics required that each of LANL, LLNL, and Sandia National Laboratory (SNL) be explicitly represented. We also insisted the codes selected be truly representative of the workloads in the labs and not be selected to make statements with respect to the performance NNSA achieves. In the end, the following choices were made. UMT2000[3] was chosen as it was developed at LLNL and models a radiation transport code. SAGE[4] was selected as it is an Eulerian hydrodynamics code from LANL. CTH[5] was selected as not only because it is a shock physics code from SNL, but because it's also the most highly utilized code at the Army Research Laboratory's Aberdeen Proving Grounds, where engineers are designing the Future Combat System.

The above work focused on applications for which the Defense community, either the government itself, or its contractors, develop their own codes. However, much of the government's workload consists of running commercial codes developed by independent software vendors (ISVs). Historically, the most important unclassified user community was mechanical computer aided engineering (MCAE), the principle code was NASTRAN[6], and the Linpack benchmark was a good proxy for it. Unfortunately, as a new generation of distributed memory computers emerged, the Linpack benchmark evolved away from the MCAE codes, and performance measured

by the benchmark no longer reflected that provided to the end users of the MCAE codes.

To address this shortfall and create a benchmark that measures the throughput of a real code, we proposed adding LS-DYNA, the most important commercial MCAE code in terms of usage today, to the HPCS full applications benchmarks. Going even further, we engaged UCSD to create a new Web site at www.topcrunch.org to track performance of LS_DYNA on different systems. TopCrunch was modeled on the TOP500 web site (www.top500.org) that tracks performance on the Linpack benchmark. Like TOP500, UCSD's plan was to pose a problem, allow computer vendors to have their MCAE marketing engineers optimize and run the LS-DYNA code, and then for UCSD to collect and publish the results. Two problems were posted, and results have been collected and posted from a variety of MCAE users as well as computer vendors. Further details of this work can be found in Appendix C, which contains a brief report from UCSD.

One of the reasons the Linpack benchmark gained a life of its own, independent of any relationship to mainstream applications, was that it was used by both computer vendors and users for bragging rights. This meant that over the last decade, these people invested tremendous effort to port the code and optimize it. The TOP500 Web site merely has to establish the rules and publish the results. This too is a goal of the TopCrunch benchmark, to stimulate enough interest that it becomes self-sustaining and outlives the HPCS program.

In order to represent the needs of a broader set of the intelligence community our team studied the data flows and algorithms used in specific intelligence systems. Further we developed a characterization table for future systems based on: process type, products, functions, data size, and throughput. This work culminated in a series of one-day workshops that were held for each of the HPCS vendor teams. These workshops were classified, and no information will be conveyed in this report.

Our team member NGC examined how to represent intelligence processing requirements to the HPCS vendors. They reported that such codes are typically written in C or C++ by teams of up to fifteen people and have a lifetime of up to twenty years. An abstract algorithm that could be representative of this class of computation is set partitioning under constraints including non-linear least squares. Northrup-Grumman outlined a benchmark that would implement this algorithm. The proposed benchmark is discussed in further detailed in Appendix H, a report by NGC.

Developers of embedded HPC systems are increasingly considering tiled architectures, in which a relatively simple processing component is replicated many times in an array. While such systems provide very high compute density, their system architecture is unusual and thus their programmability is open to question. Researchers at ISI in Arlington, VA, studied the question of mapping a common algorithm, an FFT, onto an example of such a tiled array, the MIT RAW[7] system. They reported that the Static Communication API was able to profile the communication activity in the algorithm

and automatically determine a static routing pattern. This work suggested that tiled arrays may offer reasonable levels of programmer productivity for Defense systems. Appendix B provides further detail of this work.

Productivity is the ultimate theme of the HPCS program. Each of the HPCS vendor teams was expected to develop its own definition of productivity and share it with the HPCS community. The goal was to develop a consensus within the Defense HPC community. In the end, all vendor teams produced formulas for productivity as the ratio of utility divided by cost. Utility is a complex function of the timeliness of results. Cost is the total cost of ownership, including the labor to develop codes, the purchase price of the system itself, and the cost of maintaining the system for over its operation lifetime. A special issue of the International Journal of High Performance Computing Applications (Volume 18, Number 4, Winter 2004) was published in November 2004 containing papers generated by HPCS program investigators and edited by Lincoln Lab's Dr. Jeremy Kepner.

Our role in defining productivity was to help facilitate the process. Towards that end, we organized a workshop in Santa Monica, CA in January of 2004. Appendix A contains the agenda for the January 2004 HPCS Productivity Workshop.

4. HPCS Supplementary Effort

To help facilitate the transition of the HPCS program from phase one to phase two, the project was extended so we could continue to collaborate with MITRE on the HPCS benchmarking activity and to create a Web site for the HPCS community. However, the focus of the second phase of the HPCS Analysis effort was on planning and initiating two projects whose goals would be to develop methodologies for quantifying the productivity of future HPCS systems. This work was partitioned into two subsets, development time to reflect the human cost of writing HPC code, and execution time to reflect the throughput of the codes on HPC systems. The bulk of the effort was directed at development time as there is no prior art or related work being pursued elsewhere. Each of the four foci of the supplementary project is addressed in turn below.

4.1 Benchmarking

ISI continued its successful collaboration with MITRE to organize the HPCS benchmark set. A revised four-tiered HPCS benchmark strategy was developed, as depicted in Figure 1. On the left are the HPCchallenge benchmarks (www.hpcchallenge.org). The overall Petascale performance goals of the HPCS program correlate to the throughput measured by the High Performance Linpack (HPL), parallel RandomAccess, and PTRANS kernels of HPCchallenge. The next box to the right represents the HPCS kernel benchmarks, largely unchanged from the first phase of HPCS. Continuing to the right, the next box represents a new set of scalable synthetic compact applications (SSCA). These are intended to provide HPCS scientists and engineers with model applications that are of a scale that they can be easily reimplemented in different programming languages, and for a variety of systems, enabling the community to measure the impact of these technologies on productivity. Finally, the rightmost ovals depict the representative mission partner applications. At the end of the HPCS Analysis project, HPCchallenge was up and running, MITRE was distributing the kernel benchmarks and mission partner applications, and Lincoln Labs was developing the scalable synthetic compact applications.

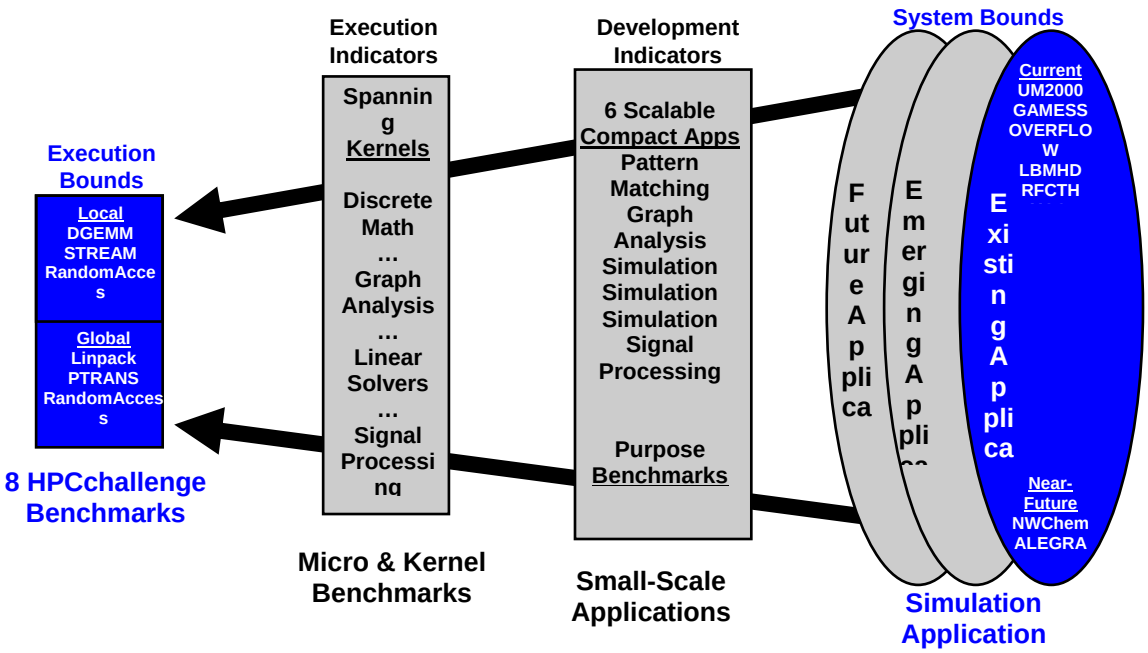

Figure 1: The four-tiered HPCS benchmarking hierarchy.

The HPCchallenge was created to augment the famous High Performance Linpack benchmark used in the TOP500 with kernels exhibiting more challenging memory access patterns. The HPCchallenge benchmark is managed by the Univerisity of Tennessee at Knoxville (UTK) via the web site www.hpcchallenge.org. Both source code for the benchmark and results can be found at the web site. One of the HPCchallenge codes, RandomAccess, is derived from the HPCS discrete math benchmarks that we released, and thus we have the role of maintaining this code. We

also continue to maintain and distribute the other HPCS discrete math benchmarks as needed.

The kernel benchmarks remain largely unchanged from the first phase of HPCS. They were augmented with the addition of the STREAMS benchmarks and the other components of the HPCchallenge benchmark. In addition, it was proposed that a new set of I/O benchmarks be developed to better represent the needs of the intelligence community to work with both very large data sets in secondary storage as well as high-speed streams of raw data from sensor platforms.

The scalable synthetic compact applications were designed to provide HPCS scientists with model applications that were more complicated than the kernel benchmarks, yet could still be reimplemented in different programming languages. Six such benchmarks were proposed. The first three SSCAs were selected to be related to otherwise classified Defense problems. One would encompass the graph analysis kernels, a second would involve pattern matching, and a third signal processing. Three additional simulation SSCAs were also proposed. They would be representative of applications in chemistry, adaptive mesh refinement, and multi-physics codes. At the end of the HPCS Analysis project, the graph analysis SSCA and been specified, and Lincoln Labs had begun implementing it. Development and release of the rest of them was deferred until later in HPCS phase two.

SUN, one of the three HPCS phase two system vendors also proposed a set of so called purpose-based benchmarks. These are similar in scale to the SSCAs. However, they differ in that rather than specifying an abstract mathematical problem, they instead pose a representative problem from an application domain, leaving the solution technique entirely up to the programmer. At the end of the HPCS Analysis project, SUN had proposed nearly ten such purpose benchmarks, and had made significant progress on the first, taken from the mechanical engineering domain.

The choice of representative HPCS mission partner codes was revisited during the supplementary phase of the HPCS Analysis project. The SAGE code was dropped because it turns out to have components that are proprietary to the Science Applications International Corporation (SAIC). Furthermore, it became necessary to include codes sponsored by a number of mission partners including DOE SC, NSF, and NASA. ISI and MITRE insisted that the codes span a wide application space, as defined by the HPCMP's computational technology areas (CTA). Finally, source code had to be available and there could be no restrictions on access to the code by foreign nationals.

Table 3 contains the set of representative mission partner applications as of the end of the HPCS Analysis project. UMT2000 and CTH were carried over from the first set of applications. OVERFLOW-D is a computational fluid dynamics (CFD) code developed by NASA and used by the Army to study rotorcraft. GAMESS and NWChem are computational chemistry codes. ALEGRA is another Sandia shock physics code, whose presence in this collection was requested by NNSA. LBMHD and GTC are magnetic confinement fusion codes. HYCOM is a new Navy ocean modeling

code. Finally, the Community Climate System Model (CCSM) is the World standard code for modeling long-range climate change. Note that this set not only covers a wide range of computational disciplines, but it also was chosen such that each mission partner was explicitly represented by a code it invests in.

Application	CTA	Mission Partners				
		DoD HPCMP	DoE OoS	DoE NNSA	NASA	NSF
OVERFLOW-D (2)	CFD					
GAMESS	CCM					
NWChem	CCM					
UMT2000	RT					
RF-CTH	CSM					
ALEGRA	CSM					
LBMHD/GTC	CEA					
HYCOM	CWO					
Community Climate System Model (CCSM)	CWO					

Table 3: HPCS Mission Partner representative applications

UCSD also continued to maintain and enhance the TopCrunch, full application benchmark and web site. In the course of this work, UCSD invented a strategy for making arbitrarily large models by creating chain-reaction car crashes, allowing the benchmark to scale to arbitrarily large problems in the future. A Web site such as TopCrunch can only be successful long term if the computer vendors compete to perform the best. The TopCrunch Web site is well on its way to providing a full application alternative to the TOP500. Again, additional details are provided in Appendix C, which contains a report from UCSD.

Ohio State University (OSU) prototyped an environment for automatically specifying and executing a set of benchmarks. This is needed for the second phase of HPCS, where users are expected to want to run a wide variety of HPCS benchmarks, implemented in different programming languages, and a variety of platforms. The number of alternatives is such that automation of this process is necessary lest the productivity of HPCS researchers suffer.

4.2 HPCS Community Web Site

ISI developed a web site to allow for the dissemination of information and the easy exchange of data. The address for the site is www.highproducivity.org. It contains publicly available information such as an introduction to the HPCS program, meeting announcements, and a list of participants. There is also a password protected area which enables HPCS working groups to share documents. There are email aliases to

facilitate communication amongst the members of the productivity team. Finally, there is a link to the HPCchallenge benchmarks which are maintained on the University of Tennessee's web site, www.hpcchallenge.org. After prototyping the site for the HPCS program, We turned it over to the Georgia Institute of Technology for long-term maintenance. The ultimate goal for this web site is that it will outlive the HPCS program and become an enduring resource for the Defense HPC community.

4.3 Development Time

The HPCS phase two development time project aims at the analysis of HPC technologies to evaluate them with respect to development-time tradeoffs as well as the analysis of HPC technologies to understand them with respect to common software development. In order to achieve these goals HPCS researchers must characterize and understand individual HPC technologies with respect to attributes such as ease of use, ease of learning, and types of defects, for a particular set of context variables. Different metrics and models exist for measuring and predicting execution time under various conditions. However, little empirical study has been done on the human effort required to implement those solutions. As a result, many development decisions about language and approach are made based on anecdote, "rules of thumb," or personal preference. Without empirical data, governmental organizations cannot take into account the effects that an HPC system will have on the development time of Defense applications. Such data is necessary to help guide decision-making when purchasing the next generation of HPC systems. Therefore, UMD set out to initiate an empirical study of HPC software development for the second phase of the HPCS program.

Understanding a discipline involves generating hypotheses, building models (e.g. application domain, workflows, problem solving processes), checking whether the understanding is correct (e.g. testing the models and experimenting in the real world), and finally analyzing the results to learn, encapsulate knowledge, refine models and evolve hypotheses. The first step towards understanding a discipline is the definition of various variables and factors involved. In the context of HPCS development time, three main types of variables can be recognized: *controlled independent variables*; *non-controlled independent variables*; and *dependent variables*. Controlled independent variables are factors whose effects are to be studied and manipulated in an experiment. Examples of these variables are problem type (e.g., embarrassingly parallel or nearest-neighbor), problem domain (e.g., weather simulation or image processing), program size (e.g., kernel or compact application), hardware (e.g., cluster), programming model (e.g., message passing or shared memory), implementation of programming model (e.g., MPI), base programming language (e.g., C, Fortran or Matlab), access to existing serial implementation and software development process. Non-controlled independent variables are factors which the experimenter cannot usually control but may have an effect on dependent variables. For HPC, these context variables are usually human attributes such as experience, knowledge of problem domain, educational major (e.g., physics, computer science, engineering, or math), relevant courses taken, motivation and inherent programming ability. Dependent variables are factors which are supposed to be explained by changes in independent variables. Variables such as execution time,

speedup relative to serial implementation, development time (total or breakdown by activity) and source lines of code (SLOC) are dependent.

The development time effort's plan is to conduct human subject software development experiments in three different contexts. The first is the classroom, which affords inexpensive access to a large number of subjects, and was the focus of almost all of the effort in this initial phase. The second is an industrial context, involving software development professionals. The third is an observational study, in which one analyses the data collection tools. UMD was also able to begin analyzing some of the preliminary data collected in pilot classroom experiments.

Classroom Studies

Classroom studies are thought to be representative of a subset of the Defense HPC software development environment similar to that of the "lone researcher". The results obtained from these studies could be generalized for scientists who need to do HPC programming, but do not have HPC experience. Through classroom studies we are able to measure ease-of-learning, provide evidence for or against "tribal lore", and relate the results to the kernels developed by the Benchmarking Working Group. Other advantages of classroom studies are running experiments with more subjects, less cost, and faster results. Classroom studies also allow researchers to analyze the effects of variables which may be impossible to control on a larger project, identify potentially statistically significant relationships among a large number of variables, and debug the experimental protocol before applying it to more expensive studies involving professionals.

Pilot classroom studies to test and validate the classroom methodology were conducted by Jeff Hollingsworth at UMD, Alan Edelman at the Massachusetts Institute of Technology (MIT), John Gilbert at the University of California at Santa Barbara (UCSB), Allan Snavely at UCSD, and Mary Hall at the University of Southern California (USC). The San Diego Supercomputing Center (SDSC) provided a computing platform and the UMD provided an exemplar experimental software suite as well as guidance on the experimental procedures to the investigators. UMD also helped each of the classroom instructors work through their respective institution's procedures for authorizing research involving human subjects (i.e., the students). Table 1 contains the classroom study schedule.

Study	Period	Location	Status
Pre-Pilot Study	Fall 2003	University of Maryland	Completed –Analyzed
Pilot Studies	Spring 2004	MIT, USC, UCSB, UMD	Completed –Under analysis
Pilot Study	Fall 2004	UCSD	TBD
Full Study	Spring 2005	TBD	TBD

Table 4: HPCS Classroom studies schedule

The classroom studies were run in 6 classes, 13 assignments were given and data from 100 subjects was captured, 71 background questionnaires, 41 post-test questionnaires and about 500 effort-log entries were collected. Nearly 1,500 hours of effort was reported, 26,000 time-stamped source files with activity, 2,600,000 lines of code (multiple versions of each file) and 16,000 time-stamped execution runs were captured.

Through these classroom pilot-studies, good collaboration with all professors involved was established, software engineering empiricists at UMD learned about the HPCS program, and HPC professors learned about human-subjects experimentation. A rich set of hypotheses was generated, a web-based experience base was created, automatic data collection mechanisms were established, and good initial data on variables such as source code, different development activities, and the time spent during each of these activities was captured.

Valuable lessons were learned about HPC infrastructure and both automated and manual data collection. Technical difficulties are common on HPC machines and it is not always possible to re-run the code on the same machine later on. The data collection process is affected by students' access to un-instrumented machines and frequent questions on instrumented machines may result in inaccurate answers. Furthermore, the experimentalists must set up data collection mechanisms, otherwise important data may not be collected. Reported effort logs are unreliable and the best way to get subjects to complete questionnaires is to be in the room as they fill them out.

Additional details about UMD's pioneering work in empirical studies of HPC software development can be found in Appendix D. Reports from UCSD, UCSB, and MIT regarding their development time classroom experiments can be found in Appendices E, F, and G.

Industrial Studies

The first industrial case study was the implementation of the Graph Analysis executable specification, an HPCS scalable synthetic compact application drafted at Lincoln Labs. The goal is to develop a compact application that has multiple kernels accessing a single data structure representing a directed multi-graph, with weighted edges. The problem includes the implementation of four computational kernels: Graph Construction, Sort on Large Sets, Graph Extraction and Graph Clustering. At the end of the HPCS Analysis project, this activity was not yet complete. It will be completed as part of the Lincoln Labs and UMD HPCS phase two efforts. Additional industrial HPCS development time experiments are also being planned.

Observational Studies

Finally, in order to evaluate development time experimentation mechanisms and tools, and compare the results obtained from various mechanisms, UMD designed controlled observational studies. In these studies one observes the developer throughout the software development process and collects the development data in several ways. By comparing the results one can evaluate the effectiveness of data collecting methods

such as manual data collection, Eclipse[8], Hackystat[9] and other instrumentation mechanisms. This work will be conducted later in HPCS phase two.

Data Analysis

Analysis of collected data from pilot classroom studies is underway. Although the data is not yet validated, a rich set of hypotheses has emerged from HPCS course professors' observations, the data, and experts in HPC community. Rather than creating new hypotheses and evaluating them, UMD collected "Tribal Lore" from the HPC community and tried to evaluate its validity by using the experimental data. Three examples are given below:

The following hypothesis was suggested by Mary Hall (instructor of USC595 pilot study): Performance tuning of uniprocessor code (optimizing for cache and registers) takes a substantial fraction of the overall tuning effort.

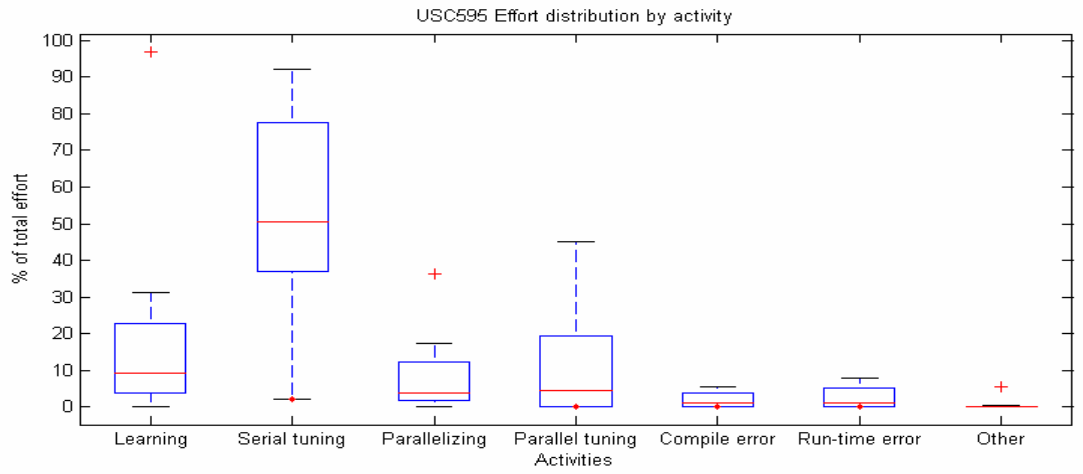

Figure 2: Effort percentage spent on each activity (USC595)

Figure 2 represents the effort students spent on various activities for the study USC595. As seen in the figure, the collected data approves the hypothesis.

Another hypothesis, common to HPC practitioners is: The variation in the speedup of MPI codes will increase with the number of processors

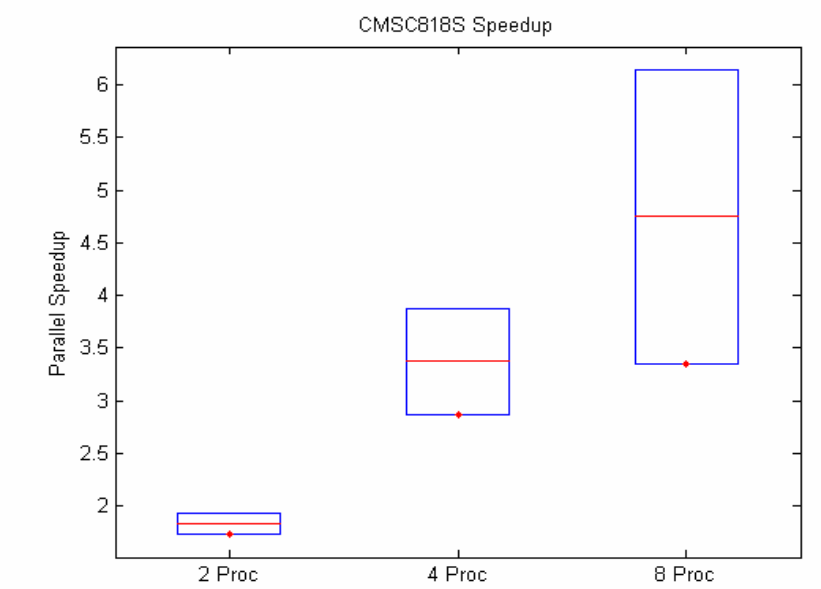

Figure 3: Evidence for hypothesis2 from pilot study CMSC818S

Figure 3 represents the range of parallel speedup observed in data from the CMSC818S class. As seen in the figure, the collected data again approves the hypothesis.

The following hypothesis emerged from collected data: There is a difference between people who develop their code at a steady pace and those who "panic" at the deadline in terms of: code size, total effort, code performance, and defects. Figure 4 represents the data that this hypothesis emerged from. Notice that the cumulative effort for the student who worked at a steady pace was less than half that of the student who "panicked" and finished the project the night before it was due.

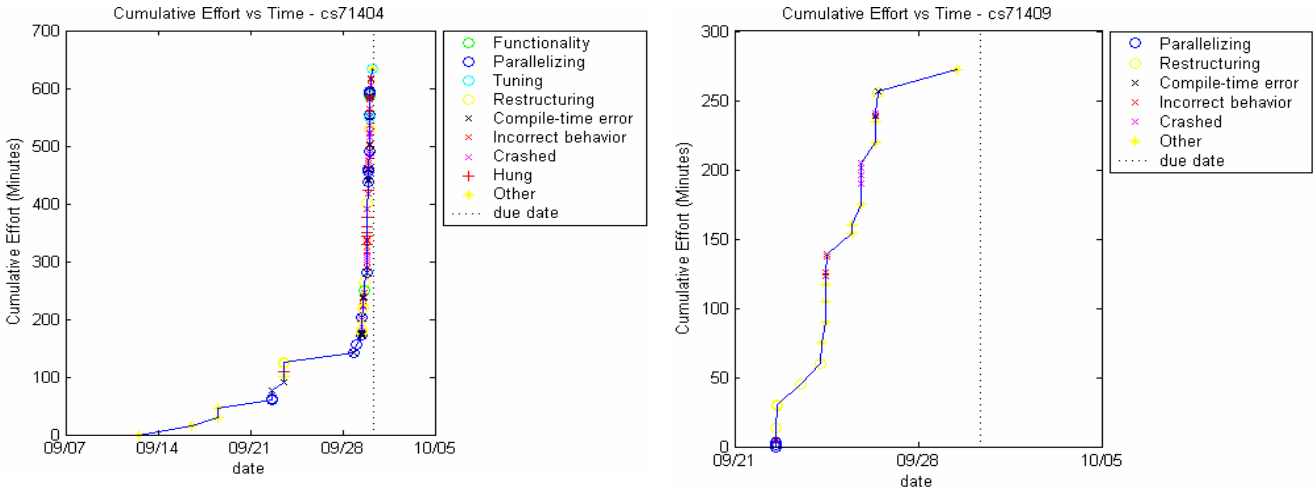

Figure 4: Steady Development (left) versus Panic (right)

4.4 Execution Time

The HPCS phase two execution time plan was developed by ISI in coordination with LLNL and UCSD. An overview is presented below. Due to the fact that related performance work was ongoing in the broader HPC community, yet there was no analog to the development time effort, DARPA chose to defer initiation of the execution time plan until FY04 funding from a mission partner, DOE SC, became available. Thus only a plan, but no early results are presented here.

The HPCS execution time research project was designed to deliver improved understanding of architectural factors affecting application execution time on 2010, Petascale systems. The initial goal is to establish a baseline by understanding the critical performance features on existing cutting-edge systems, such as the Cray X1[10], then move to near-term future systems, such as Red Storm and BlueGene/L[11], as they arrive. We then plan to develop tools for creating models of future applications by scaling from contemporary problems. The execution time of these model Petascale applications would then be estimated by running them on simulators for future machines with new architectural features. To facilitate these studies we propose developing a Common Modeling API allowing for graceful connection to other HPCS measurement, modeling, and simulation tools. Finally, we plan to work closely with the HPCS program's MITRE team on representative Defense codes and compact applications to ensure we model representatives from different dimensions of the HPCS application space.

The execution time plan is structured around five primary thrusts:

1. Study application execution on existing systems to identify critical performance features.
2. Develop scaling models of applications to predict future machine requirements.
3. Develop a scalable system modeling capability for estimating application performance given key system parameters of future architectures.
4. Connect tools for measurement, modeling, and simulation with a common modeling API.
5. Predict end-user productivity when developing and executing full-scale applications.

The foundation on which all of the rest of this system builds will be a set of performance tools that use both static analysis of source code and traces of run-time behavior to extract detailed information about how and why an application performs as it does on a particular computing system. We will integrate the outputs of a variety of tools, both those we are developing as part of our existing research as well as research and commercial tools such as Tau[12] and Vampir[13] which are familiar to us. Our static analysis will allow us to generate performance assertions to identify potential performance bottlenecks and focus our analysis. Our trace tools will determine specifically what the bottlenecks are. We will be able to quantify performance for

codes today, setting realistic expectations for Petascale systems at the end of the decade.

A detailed breakdown of the software methodology for scaling applications and modeling their performance on Petascale systems is illustrated in Figure 5. Each of the software modules and interface specifications to be developed are shown explicitly. Specific parallel kernels from the HPCS benchmarks will be extracted to provide a test suite for the execution evaluation experiments. Trace tools will allow detailed analysis of executing code to establish a directed graph representing the unfolding computation and identifying the local computation, memory access patterns, remote request messages, and global synchronization such as barriers. This provides a quantitative means of evaluating the key operational properties of the application driven computation. The result will be a specification and description of the resulting computational workload in terms of concurrency, precedent constraints, and classes of operations.

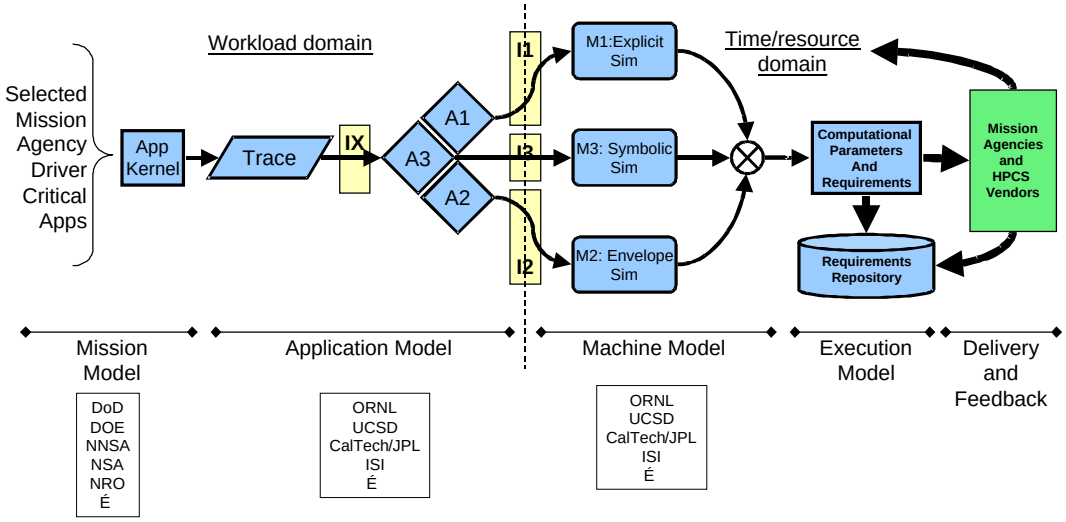

Figure 5: Detailed Tool Flow for Application Scaling and System Modeling.

The Petascale HPCS platforms to be considered will be vastly larger than existing machines and the workloads to be run on them are anticipated to be many orders of magnitude greater than those extracted and characterized from the kernels by the trace tool suite. Parameterized scaling models (A1 – A3) will be developed for each application kernel. Initially this will be done by hand and is expected to be labor intensive. The scaling model will relate the primary characteristics of the workload to a set of scaling parameters. An open interface format will be defined and include symbolic parameters and symbolic relationships (I1 – I3). The scaling model accepts the scaling relationships and transforms the measured values of the trace workload to produce a new workload characterization.

We proposed to implement four models of machines to examine different operational properties and performance tradeoffs. The machine models will fall in to different

categories depending on the form of workload scaling model used. An explicit thread machine model will accept a description of an abstract machine at scale and the representation of the explicit thread scaled workload. It will apply a transformation to convert the workload data in to a time domain based representation of the machines operational characteristics. The UCSD Metasim Convolver[14] tool will be extended to input trace-derived threads and feed them to a discrete event simulation of a computing system. A continuum envelope machine model will be developed that takes an input workload envelope and creates a time domain output workload. Because some of the precedent constraint information will be lost in the envelope representation, this machine model will generate two new outputs, one for eager evaluation which will produce a lower bound on execution time, and one for lazy evaluation which will produce an upper bound on execution time. The final machine model, the symbolic model, will work directly on a symbolic workload representation and keep some of its parameters to produce a symbolic representation of the time domain behavior of the modeled machine. This will allow symbolic manipulation of the results to compute such higher order properties as sensitivities with respect to key parameters.

When successful, the HPCS execution time project will demonstrate for the HPC community the ability to extrapolate an existing application's performance characteristics to represent a future workload. These models will then be studied with simulations of future systems to anticipate the execution time performance such systems will provide to their users. They could also be used to define future HPC requirements.

5. Ongoing and Future Work

The DARPA HPCS program is ongoing as this report is written, and the productivity team that ISI helped create continues to play a vital role. A critical aspect of this is a rich set of benchmarks and applications that was developed to represent the needs of Defense science and engineering workload to the HPCS vendors and the broader HPC community. ISI continues to participate in the HPCS benchmarking work, in collaboration with MITRE. The most visible aspect of the HPCS benchmarking effort is the HPCchallenge benchmark which augments the TOP500 with code kernels exhibiting more challenging memory access patterns. The HPCchallenge benchmark is maintained by University of Tennessee at Knoxville (UTK). There is also an ongoing effort led by OSU to develop a test and specification software infrastructure to facilitate benchmarking with a wide variety of kernels, programming models, and systems.

Another productivity team project is underway to learn how to quantify the relative costs of different programming models in the development of Defense applications. Led by the UMD, an empirical study of programmer productivity is being developed. This is being complemented by a study of existing, large-scale scientific and engineering code projects. The existing codes study is being led by LANL's Dr. Douglass Post. The ultimate goal of these is to provide a quantitative basis by which Defense program managers can anticipate HPC software development costs and use this information as part of overall system procurement strategies.

Just as it's important to anticipate software development costs, so too is it important to be able to anticipate how well future HPC systems will perform, even before the first prototypes exist. This is because the Defense community is usually the pioneering user of such systems. Therefore, an execution time project led by ISI has also been initiated to model current Defense applications at the Petascale, study their behavior on models of HPCS systems, and enable engineers to reason about architectural tradeoffs.

6. Publications

Jeff Carver, Sima Asgari, Victor Basili, Lorin Hochstein, Jeffrey Hollingsworth, Forrest Shull, and Marv Zelkowitz. "Studying Code Development for High Performance Computing: The HPCS Program." In Proceedings of the Workshop on Software Engineering and High Performance Computing Applications (held at ICSE 2004). Edinburgh, Scotland.

Sima Asgari, Victor Basili, Jeff Carver, Lorin Hochstein, Jeffrey Hollingsworth, Forrest Shull, and Marv Zelkowitz. "Challenges in Measuring HPCS Learner Productivity in an Age of Ubiquitous Computing." In Proceedings of the Workshop on Software Engineering and High Performance Computing Applications (held at ICSE 2004). Edinburgh, Scotland.

Victor Basili, Sima Asgari, Jeff Carver, Lorin Hochstein, Jeffrey K. Hollingsworth, Forrest, Shull, Marv Zelkowitz. A Pilot Study to Evaluate Development Effort for High Performance Computing." University of Maryland Technical Report CS-TR-4588. April 2004.

Sima Asgari, Victor Basili, Jeffrey Carver, Lorin Hochstein, Jeffrey K. Hollingsworth, Forrest Shull, Marvin Zelkowitz. " High Productivity Computer Systems (HPCS): Empirical studies on Development Time" In the poster session of the 2004 ACM-IEEE International Symposium on Empirical Software Engineering (ISESE 2004) 19-20 August 2004 Redondo Beach CA, USA

Ron Choy, Alan Edelman, John R. Gilbert, Viral Shah, and David Cheng. "Star-P: High productivity parallel computing." Accepted to Eighth Annual Workshop on High-Performance Embedded Computing (HPEC-04).

John R. Gilbert and Viral Shah. "Graphs and sparse matrices in an interactive supercomputing environment." Minisymposium talk presented at SIAM Annual Meeting, 2004.

Imran Patel and John R. Gilbert. "Grid*P: Interactive supercomputing on the grid with Matlab." Presented at SIAM Conference on Parallel Processing for Scientific Computing, 2004.

Viral Shah and John R. Gilbert. "Sparse matrices in Matlab*P: Design and implementation." Accepted to 11th Annual International Conference on High Performance Computing (HiPC 2004).

Ron Choy, Alan Edelman, John R. Gilbert, Viral Shah, and David Cheng. "Star-P: High productivity parallel computing." Accepted to Eighth Annual Workshop on High-Performance Embedded Computing (HPEC-04).

7. Personnel

7.1 ISI Research Staff:

- Dr. Robert Lucas, Principal Investigator
- Dr. Pedro Diniz
- Dr. Mary Hall
- Dr. Jacqueline Chame
- Mr. Daniel Davis
- Mr. Spudun Bhatt
- Dr. Barbara Yoon (consultant)

7.2 Subcontractor Research Staff

- MIT: Alan Edelman
- NGC: Robert Babb
- NGC: Mark Coffey
- OSU: Ashok Krishnamurthy
- SGI: William Harrod
- UCSB: John Gilbert
- UCSD: Allan Snavely
- UCSD: Dave Benson
- UMD: Vic Basili
- UMD: Jeff Hollingsworth

8. Results, Conclusions & Technology Transfer

The HPCS Analysis project supported DARPA's HPCS effort, allowing ISI to work with MITRE and Lincoln Labs to represent Defense computing needs to the broader HPC community, to develop the HPCS benchmarking suite, prototype the www.highproductivity.org web site, and to develop methodologies for measuring both the development as well as the execution time productivity of Defense HPC systems and their programming environments. In doing so it achieved all of the goals set for it in the HPCS Analysis project and its supplement.

There are some early results to report from the HPCS Analysis project. The HPCS benchmarks and the HPCchallenge benchmark were developed. We released the HPCS discrete mathematics benchmarks to the HPC community and maintain the RandomAccess code. We initialized the www.highproductivity.org web site and transitioned its maintenance to Georgia Tech. We planned and organized the HPCS phase two development time and execution time activities. Furthermore, our team even did some pioneering work, performing development time experiments in HPC classrooms.

The HPCS benchmarks, HPCchallenge, and TopCrunch are already in use by the broader HPC and MCAE communities. The productivity methodologies have been published, but not yet incorporated into any government procurements at this time. The execution and development time research projects are too premature to have produced any definitive conclusions.

9. Inventions, or patent disclosures

No inventions were disclosed or patents submitted by the ISI HPCS Analysis research team or its subcontractors.

10. References

[1] Jack Dongarra, “Performance of Various Computers Using Standard Linear Equations Software”, (Linpack Benchmark Report), University of Tennessee Computer Science Technical Report, CS-89-85,2004

[2] Souli, M. Sofiane, Y. Olovsson, Lars. “ALE and fluid/structure interaction in LS-DYNA”, American Society of Mechanical Engineers, Pressure Vessels and Piping Division, (PVP) Emerging Technology in Fluids, Structures, and Fluid-Structure Interactions, v 485 n1, 2004. p 181-187

[3] J.S. Vetter and A. Yoo, “An Empirical Performance Evaluation of Scalable Scientific Applications,” Proc. SC 2002, 2002.

[4] Gisler, Galen R. Weaver, Robert P. Mader, Charles L. Gittings, Michael L. “Two- and three-dimensional asteroid impact simulations”, Computing in Science & Engineering. v 6 n 3 May/June 2004. p 46-55

[5] E.S. Hertel, Jr, R.L. Bell *et al.*, “CTH: A Software Family for Multi-Dimensional Shock Physics Analysis,” Proc. Proceedings of the 19th International Symposium on Shock Waves, 1993, pp. 377-82.

[6] Cote, F. Masson, P. Mrad, N. Cotoni, V., “Dynamic and static modelling of piezoelectric composite structures using a thermal analogy with MSC/NASTRAN”, Composite Structures, v 65 n 3-4, September 2004, p 471-484

[7] Anant Agarwal, et.al., "Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams”, Proceedings of International Symposium on Computer Architecture, June 2004.

[8] Sherry Shavor , Jim D'Anjou , Pat McCarthy , John Kellerman , and Scott Fairbrother, “The Java Developer's Guide to Eclipse”, , Pearson Education, 2003

[9] Philip M. Johnson, “You can't even ask them to push a button: Toward ubiquitous, developer-centric, empirical software engineering”, The NSF Workshop for New Visions for Software Design and Productivity: Research and Applications, Nashville, TN, December, 2001.

[10] Steven Vaughan-Nichols, “New Trends Revive Supercomputer Industry”, IEEE Computer, v37 n2, Feb 2004

[11] George Almasi, *et.al.*, “Unlocking the Performance of the BlueGene/L Supercomputer”, Proc. SC2004 , November 2004

[12] Sameer Shende , Allen D. Malony, “Integration and applications of the TAU performance system in parallel Java environments”, Proceedings of the 2001 joint

ACM-ISCOPE conference on Java Grande, p.87-96, June 2001, Palo Alto, California, United States

[13] Pallas GMbH, "VAMPIR: Visualization and Analysis of MPI Resources", http://www.pallas.de/pages/vampir.htm

[14] Laura Carrington, Nicole Wolter, Allan Snavely, and Cynthia Bailey Lee "Applying an Automated Framework to Produce Accurate Blind Performance Predictions of Full-Scale HPC Applications", UGC 2004, Williamsburgh, June 2004.

11. List of Acronyms

API	Application Programming Interface
ASCI	Accelerated Strategic Computing Initiative
BMM	Bit-Matrix-Multiply
CCS	Center for Computing Sciences
CCSM	Community Climate System Model
CTA	Computational Technology Area
DARPA	Defense Advanced Research Projects Agency
DOD	Department of Defense
DOE	Department of Energy
FFRDC	Federally Funded Research and Development Center
FFT	Fast Fourier Transform
GF(2)	Galois Field Two (i.e., binary arithmetic)
HPC	High Performance Computing
HPCS	High Productivity Computing Systems
HPCMP	High Performance Computing Modernization Program
HP	Hewlett Packard Corporation
HPL	High Performance Linpack
IBM	International Business Machines Corporation
ISI	Information Sciences Institute
ISV	Independent Software Vendor
LANL	Los Alamos National Laboratory
LLNL	Lawrence Livermore National Laboratory
MCAE	Mechanical Computer Aided Engineering
MIT	Massachusetts Institute of Technology
MPI	Message Passing Interface
NAS	NASA Advanced Supercomputing Division
NASA	National Aeronautics and Space Administration
NGC	Northrup-Grumman Corporation
NNSA	National Nuclear Security Administration
NPB	NAS Parallel Benchmarks
NSA	National Security Agency
NSF	National Science Foundation
ORNL	Oak Ridge National Laboratory
OSU	Ohio State University
PI	Principal Investigator
SAIC	Science Applications International Corporation
SC	Office of Science
SLOC	Source Lines of Code
SNL	Sandia National Laboratory
SSCA	Scalable Synthetic Compact Application
UCSB	University of California at Santa Barbara
UCSD	University of California at San Diego
UMD	University of Maryland
UPC	Unified Parallel C

12. Appendices

Appendix A: January 2004 HPCS Productivity Workshop Agenda

```
Tuesday, Jan 13: Productivity Team
08:30-09:00 Breakfast
09:00-09:30 Opening Remarks (Graybill/DARPA & Johnson/DoE OoS)
09:30-10:00 Program Overview and Goals (Kepner/Lincoln)
10:00-11:30 Development Time Working Group
           -Pre-Pilot Results and Pilot Plans (Basili/UMD) [50 min]
           -Break [15 min]
           -HPC Class Overview (Gilbert/UCSB) [25 min]
11:30-12:00 Execution Time Working Group (Lucas/ISI)
12:00-01:00 Lunch
01:00-01:30 Programming Models Working Group (Lusk/ANL & Snir/UIUC))
01:30-02:30 Benchmarks Working Group
           -Overview and v0.1 Compact Apps (Koester/Mitre) [30 min]
           -HPCchallenge Results (Luszczek/UTK) [30 min]
02:30-03:00 Test & Spec Environment Working Group (Krishnamurthy/OSU)
03:00-03:15 Break
03:15-03:45 Existing Codes Analysis Working Group (Post)
03:45-04:30 Workflows, Models and Metrics Working Group (Kepner/LL)
04:45-05:30 Discussion

Wednesday, Jan 14: Productivity Team & Individual Working Groups
08:30-09:00 Breakfast
09:00-10:30 Vendor Updates and Feedback
           -Web Matrix ... (Mizell/Cray) [30 min]
           -Purpose Benchmarks ... (Votta/Sun) [30 min]
           -Productivity status update ... (Rajamony/IBM) [30 min]
10:30-10:45 Break
10:45-11:50 Partner Invited Presentations
           -Code Profiling (Davis/HPCMO & Snavely/UCSD) [45 min]
           -Council on Competitiveness [20 min]
11:50-01:00 Lunch
01:00-02:00 Roundtable Discussion and Feedback
02:00-05:00 Individual Working Group Meetings (Optional)
           -Development Time Working Group (Lead: Basili/UMD)
           -Execution Time Working Group (Lead: Lucas/ISI)
           -Existing Codes Analysis Working Group (Lead: Post/LANL)
           -Workflows, Metrics, Models Working Group (Lead:  Kepner/LL)

Thursday, Jan 15: Individual Working Group Meetings (Optional)
09:00-12:00 Benchmarking Summit (Lead: Koester/Mitre)
           -Test and Spec Working Group (Lead: Krishnamurthy/OSU)
```

Appendix B: Tiled Architecture Report from ISI East

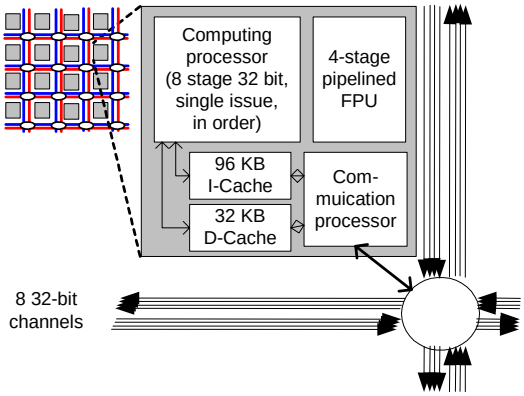

Figure 1. Raw architecture

Tiled architectures are an approach for a scalable microprocessor that addresses issues of continued technology scaling. Tiled processors implement a two-dimensional array of processors (tiles) that are connected with a mesh topology. Each tile occupies a fraction of the chip space, and clock speeds can be high since intra-processor signals only need to travel only a short distance. One example of a tiled microprocessor is the Raw chip, which has been developed and implemented at MIT. The current Raw implementation contains 16 tiles on a chip connected by a very low latency 2-D scalar mesh network. The Raw scalar mesh network is implemented with four sub-networks: two static and two dynamic. The dynamic networks require that the processor construct a header that identifies the length and destination of a message, and then messages are dynamically routed to their destination.

Messages in the static network do not require headers. Programmable routers in the static network are preprogrammed to implement the appropriate routing. These programmable routers improve performance in two ways. First, the elimination of headers improves network utilization and eliminates the software overhead required to create a header. Second, the fact that routing is determined statically at compile time allows optimized global routing to be performed. This static global optimization has the potential to increase network throughput. The disadvantage of using the static network is that inter-tile routing information must be determined at compile time. It is difficult or impossible to extract routing information from most programs written in common programming languages such as C or Java. MIT is developing a stream programming language called StreaMIT that allows a compiler to extract routing information from a program, but this requires applications to be re-written in this new language. We have developed a tool set that improves programmer

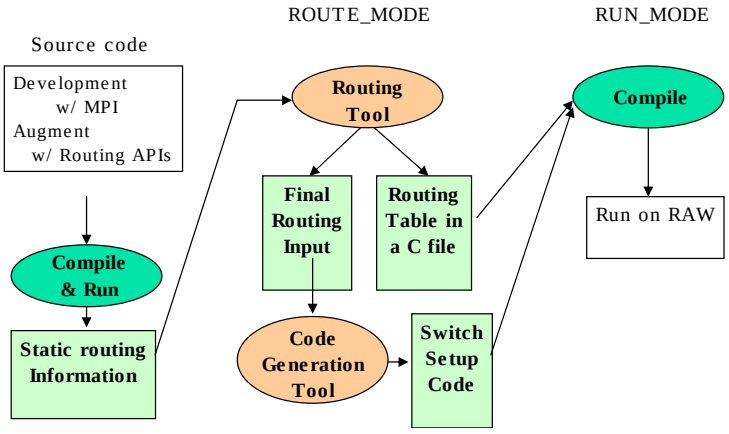

Figure 2. Static communication API tool flow

productivity for tile-based architectures by allowing minimal changes to a message-passing program written in C to be used to extract static routing information.

Our tool set requires that a parallel message passing program written in C be modified to use our Static Communication API. The Static Communication API allows the program to be run in a profile mode, during which static routing information is automatically extracted from the program. Our routing software can then be used to produce a program for the static routing network. Finally, the same program can be run in execution mode and static communication is done over the Raw static network. This tool flow is shown in Figure 2.

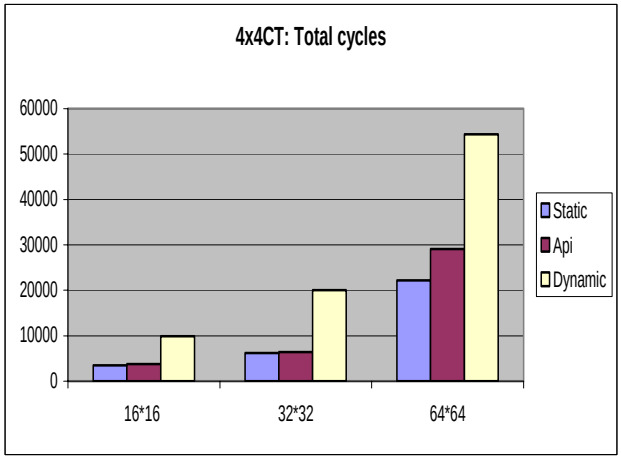

Figure 3. Corner turn performance

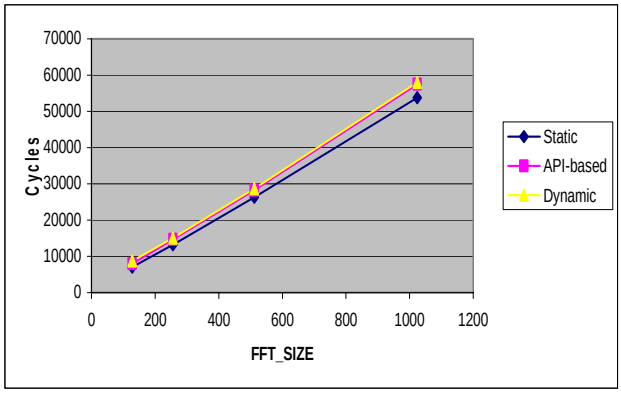

Figure 4 FFT performance

It is beyond the scope of this study to measure the productivity improvement of programmers through the use of this tool. However, our experience is that it is much easier and faster to program using this API than it is to learn the details of programming the static network. We have compared the execution performance of two application kernels using three programming method. The results are shown in Figure 3 and Figure 4. The first method is to manually program the static network. This method can achieve optimal performance (assuming the programmer has time to optimize routing and code), but is the most time consuming. The second method is to use our Static Communication API as described in the previous paragraph and shown in Figure 2. The third method is to use the dynamic network using a library based dynamic communication API. This third method is the easiest for the programmer but incurs the overheads of the dynamic network. Figure 3 shows that for smaller corner turns, our Static Communication API achieves performance roughly equal to that of the manually programmed implementation, and both static network implementations perform significantly better than the dynamic network implementation. For the largest corner turn, there is some performance degradation (about 25%) when the API is used (compared to the manual implementation), but the performance is still almost double the performance achieved using the dynamic network. The FFT performance is determined more by the

computational performance than the communication performance, so all three methodologies achieve roughly equal performance.

Appendix C: TopCrunch Report from UCSD

The TopCrunch project was initiated to track the aggregate performance trends of high performance computer systems and engineering software. Instead of using a synthetic benchmark, actual engineering software applications are used with real data and are run on high performance computer systems. The data are available for download in the form of data files for our current software suite. With time, we expect to track the evolution of delivered performance as a function of enhancements in both software algorithms and hardware. The results of the benchmarks are available as submitted, and may be searched by data, code name, and year. Series of benchmarks may also be plotted against each other.

The benchmark programs were chosen to reflect the types of calculations performed in the mechanical and aerospace communities: LS-DYNA (structural dynamics), CTH (fluid mechanics), and SPaSM (materials science). These codes have different challenges to address in terms of domain decomposition, message passing, load balancing, and dynamic memory allocation that makes the comparison of their relative scaling interesting.

The benchmark problems were chosen to reflect current engineering practice in the real world, and to have a structure that allows them to be scaled up as computer performance grows. The problems are not intended to be optimal analyses, i.e., the fastest possible choice of options to achieve a particular solution, because engineers rarely have time to optimize their analyses in real life. For example, the accuracy of the stress distribution in a structural element increases with the number of Gaussian quadrature points, but at the expense of speed. For a given level of accuracy, there is, therefore, a choice that maximizes speed. It is, however, common engineering practice to use more points than absolutely necessary because an inaccurate solution will require rerunning the analysis, which effectively doubles its cost and more than doubles the wall clock time to get an acceptable answer. The same general observation holds true for many other analysis choices to be made.

The site has been successful enough to create controversy among the participating vendors. While this has slowed the development of the site to some degree, the level of interest by both users and vendors indicates that the site will enjoy long-term success.

Three vendors, during the course of this project, questioned the accuracy of each other's benchmark results for LS-DYNA. While over 100 results have been posted during the past year, the number available to the public is 77 due to some of them being removed as being irreproducible and others being updated as performance improves. The data hasn't been destroyed for any of the results, just made invisible to the public. When we have enough data, we will study the historical evolution of the benchmark performance. The site rules required that all benchmarks be performed with production versions of LS-DYNA. Although the intent was that only the versions generally available to commercial customers could be used, some

vendors had a broader interpretation. One vendor modified the binary to enhance the performance, and another modified the source code, and neither version was generally available. Gains of up to 25% in performance were obtained for the benchmark problems.

Livermore Software Technology Corporation (LSTC), which produces LS-DYNA, initially supported tweaking the code by the vendors in the belief that the customers would benefit from the competition for better performance. Since the enhanced versions hadn't been run through the suite of test problems for quality assurance, they turned out to be less robust than the production versions. The vendors refused to disclose their modifications to the binaries and the source code because they didn't want their competitors to benefit from the enhancements. Since LSTC didn't know the nature of the changes to their own product, they were unable to support the vendor-enhanced versions. LSTC therefore withdrew support of having the benchmark results for the vendor-enhanced versions on the Top Crunch site.

An additional complicating factor was the rolling nature of the commercial releases of LS-DYNA. As new compilers and operating systems become available, and vendor-specific bugs are eliminated, LSTC releases incremental updates for individual machines. Vendors submitted results for the incremental updates prior to their appearance on the FTP site, again resulting in charges of unfair tweaking.

Several vendors sent ultimatums during the last six months threatening to withdraw their results from the site unless the Top Crunch restricted the results to only versions that are available on the LSTC web site at the time of the benchmark submission. To maintain the participation of the majority of the vendors, this has now become the standard. One vendor has withdrawn permanently.

The difficulties associated with LS-DYNA are multiplied with CTH and SPaSM since they are codes from the national laboratories that are distributed to users as source code, making it impossible to prohibit tweaking either the source or binary. Given the controversy we experienced with LS-DYNA, we are discussing with vendors the rules that should be imposed for benchmarking these codes. Progress has been slow because vendors whose machines are performing well with LS-DYNA are concerned that their machines may appear slower if other vendors aggressively modify these codes. Due to the security issues with CTH, and the small user base (in comparison with LS-DYNA), the vendors also seem less interested in benchmarking these codes. It may, in the long run, be a better strategy to restrict Top Crunch to commercial codes, and replace CTH with Fluent.

With the exception of the San Diego Supercomputer Center, obtaining benchmark results from supercomputer centers has proven difficult. Obtaining good parallel performance requires having a dedicated machine. For a major supercomputer center, these means getting them to have a system operator run the benchmarks during a scheduled maintenance period. In addition, LS-DYNA requires license keys for each processor, or a network license that requires a client program on each

processor. The level of effort required to install the licensing keys/clients on hundreds of processors has also been a major obstacle with the national supercomputer centers.

In summary, we have met our primary goal of obtaining benchmarks of actual commercial engineering software over a wide range of machines. Because of the commercial importance of our results to the vendors, whose customers recognize the limitations of using LAPACK to choose machines, we have found ourselves in the middle of controversies that we didn't anticipate. Although this initially slowed the progress of the site, the visibility gained from the controversies will ensure the project's longevity.

Appendix D: Development Time Report from UMD

Development Time Working Group Report for Phase 1

Sima Asgari[1], Vic Basili[12], Jeff Carver[1], Lorin Hochstein[1], Jeff Hollingsworth[1], Forrest Shull[2], Marvin Zelkowitz[12]
[1] University of Maryland [2] Fraunhofer Center - Maryland

July 2004

1. Introduction

The main goal of HPCS project is the analysis of vendor technologies to predict their productivity from the point of view of purchaser (government). Within the HPCS framework, the Development-time working group aims at the analysis of existing HPC technologies to evaluate them with respect to development and execution time tradeoffs and the analysis of existing HPC technologies to understand them with respect to common software development. In order to achieve these goals we must characterize and understand individual technologies by analyzing each HPC technology to understand it with respect to attributes such as development/execution time tradeoff, ease of use, ease of learning and types of defects, for a particular set of context variables. Different metrics and models exist for measuring and predicting execution time under various conditions. However, little empirical study has been done on the human effort required to implement those solutions. As a result, many development decisions about language and approach are made based on anecdote, "rules of thumb," or personal preference. Without empirical data, governmental organizations cannot take into account the effects that a high performance computing system will have on development time. Such data is necessary to help guide decision-making when purchasing the next generation of high performance computing systems.

2. Overview

Understanding a discipline involves generating hypotheses, building models (e.g. application domain, workflows, problem solving processes), checking whether our understanding is correct (e.g. testing our models, experimenting in the real world), analyzing the results to learn, encapsulate knowledge, refine models and evolve hypotheses. The first step towards understanding a discipline is the definition of various variables and factors involved. In the context of HPCS development time, three main types of variables as *controlled independent variables*; *non-controlled independent variables* and *dependent variables* can be recognized. Controlled independent variables are factors whose effects are to be studied and manipulated in an experiment. Examples of these variables are problem type (embarrassingly parallel, nearest-neighbor), problem domain (weather simulation, image processing), problem size (kernel, compact app), hardware (cluster, SMP), programming model (message passing, shared memory, mixed), implementation of programming model (MPI, OpenMP), base programming language (C,

Fortran, Matlab), access to existing serial implementation and workflow/process. Non-controlled independent variables are factors which the experimenter cannot usually control but may have an effect on dependent variables. For HPC, context variables are usually human attributes such as experience, knowledge of problem domain, major (physics, CS, EE, math), relevant courses, motivation and inherent programming ability. Dependent variables are factors which are supposed to be explained by changes in independent variables. Variables such as execution time, speedup relative to serial implementation, development time(total or breakdown by activity and SLOC are dependent.

3. Experimental Studies

In the Development-time working group we conduct three different types of experimental studies.

3.1 Classroom Studies

Classroom studies are subsets of HPC development environment similar to "lone researcher" work environment. The results obtained from these studies could be generalized scientists who need to do HPC programming but do not have HPC experience. Through classroom studies we are able to measure ease-of-learning, provide evidence for/against "tribal lore" and related the results to the kernels developed by the Benchmarking Working Group. Other advantages of classroom studies are running experiments with more subjects, less cost and faster results, studying the effects of variables which may be impossible to control on a larger project, identifying potentially statistically significant relationships among a large number of variables and debugging experimental protocol before applying it to professionals. Table 1 represents classroom study schedule.

Study	Period	Location	Status
Pre-Pilot Study	Fall 2003	University of Maryland	Completed –Analyzed
Pilot Studies	Spring 2004	MIT, USC, UCSB, UMD	Completed –Under analysis
Pilot Study	Fall 2004	UCSD	TBD
Full Study	Spring 2005	TBD	TBD

Table 1: HPCS Classroom studies

To date, the classroom studies were run in 6 classes, 13 assignments were given and data from 100 subjects was captured, 71 background questionnaires, 41 post-test questionnaires and about 500 effort-log entries were collected. Nearly 1,500 hours of effort was reported, 26,000 time-stamped source files with activity, 2,600,000 lines of code (multiple versions of each file) and 16,000 time-stamped execution runs were captured.

3.1.1 Pilot-study Successes

Through pilot-studies good collaboration with all professors involved was established, empiricists learned about HPCS and HPC professors learned about human subjects experimentation, a rich set of hypotheses was generated, a web-based experience base

was created, automatic data collection mechanisms were established. Good data on variables such as source code, different development activities and the time spent during activities was captured.

3.1.2 Pilot-study Lessons Learned

Through pilot studies we learned valuable lessons about HPC infrastructure and automated and manual data collection. Technical difficulties are common on HPC machines and it is not always possible to re-run the code on the same machine later on. Data collection process is affected by students' access to un-instrumented machines and frequent questions on instrumented machines may result in inaccurate answers. Furthermore, the experimentalists must set up data collection mechanisms, otherwise important data may not be collected. Reported effort logs are unreliable and the best way to get subjects to complete questionnaires is to be in the room as they fill them out.

3.2 Industrial case studies

Our first case study was the implementation of executable benchmark reference in MIT Lincoln Labs. The problem to be solved is HPCS Scalable Synthetic Compact Applications. The goal is to develop a compact application that has multiple analysis techniques (multiple kernels) accessing a single data structure representing a directed multi-graph with weights. The problem includes the implementation of four computational kernels: Graph Construction, Sort on Large Sets, Graph Extraction and Graph Clustering. The implementation is not yet complete. We are planning new industrial development.

3.3 Observational Studies

In order to evaluate our experimentation mechanisms and tools and compare the results obtained from various mechanisms, we are designing controlled observational studies. In these studies we observe the developer throughout the development phases and collect the development data in several ways and by comparing the results evaluate the effectiveness of data collecting methods such as, manual data collection, Eclipse, Hackystat and other instrumentations.

4 Data Analysis and Hypotheses

Analysis of collected data from Pilot studies is underway. Although the data is not yet validated, a rich set of hypotheses has emerged from HPCS course professors' observations, the data and the experts in HPC community. Other than creating new hypotheses and evaluating them, we are collecting the "Tribal Lore" from the community and trying to evaluate their validity by using the experimental data.
Sample hypotheses are given below.

Professors' Hypothesis

The following hypothesis was suggested by Mary Hall (instructor of USC595 pilot study):

Hypothesis 1: Performance tuning of uniprocessor code (optimizing for cache and registers) takes a substantial fraction of the overall tuning effort.

Figure 1 represents the effort students spent on various activities for the study USC595.

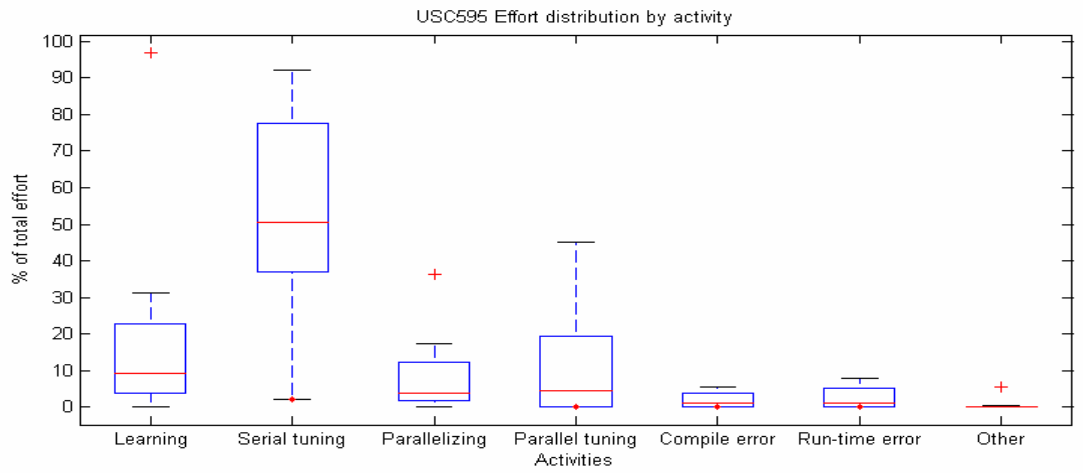

Figure 1. : Effort percentage spent on each activity (USC595)

As seen in the figure, the collected data approves the hypothesis.

Existing Hypothesis

Hypothesis 2: The variation in the speedup of MPI codes will increase with the number of processors

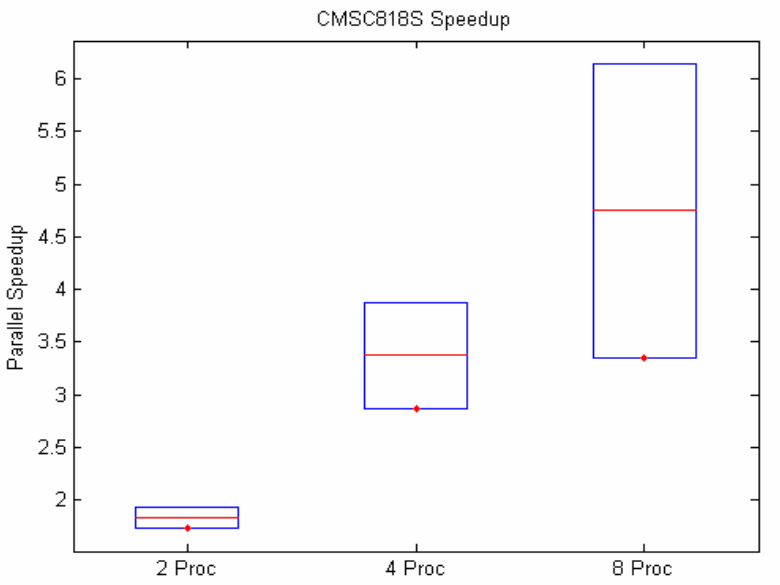

Figure 2. Evidence for hypothesis2 from pilot study CMSC818S

New Hypothesis

The following hypothesis emerged from collected data.

Hypothesis 3: There is a difference between people who develop their code at a steady pace and those who "panic" at the deadline in terms of: code size, total effort, code performance / speedup and defects.

Figure 3 represents the data that this hypothesis emerged from:

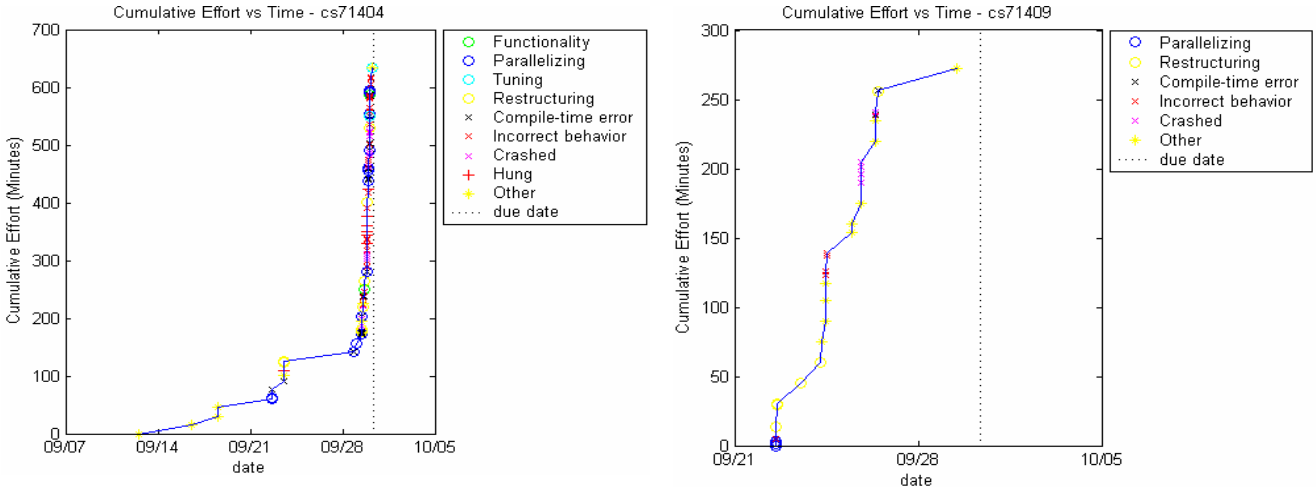

Figure 3: Steady Development (left) versus Panic (right)

5 Conclusions

5.1 Working Group Interactions

We work together with the Existing Code working group in order to integrate questions from their professional developer surveys into our classroom surveys where possible. We share our experiences and lessons learned from case studies, controlled experiments and data analysis aspects. We collaborate with the Benchmarking working group to use Kernel definitions in development time experiments and to determine which kernels are covered by existing classroom assignments. We also study development effort for Compact Applications defined by this working group. The Compact Applications could be used as basis for larger projects in HPC courses.

5.2 Future Work

We are planning to create a clearinghouse of HPCS data. Towards this goal a website of hypotheses and their supporting data has been created. More analysis will be carried out on the data by taking various context variables into account. The relationship between involved variables will be more clearly defined. Concrete metrics will be defined for each hypothesis to allow measurement and model Based on these metrics the set of hypotheses will be refined so that the hypotheses will be prioritized based on the support they receive from the data and the unsupported ones could be eliminated. We are looking to identify more professional case studies, observational studies and additional pilot studies (Fall 2004)

6 Publications

- Jeff Carver, Sima Asgari, Victor Basili, Lorin Hochstein, Jeffrey Hollingsworth, Forrest Shull, and Marv Zelkowitz. "Studying Code Development for High Performance Computing: The HPCS Program." In Proceedings of the Workshop on Software Engineering and High Performance Computing Applications (held at ICSE 2004). Edinburgh, Scotland.

- Sima Asgari, Victor Basili, Jeff Carver, Lorin Hochstein, Jeffrey Hollingsworth, Forrest Shull, and Marv Zelkowitz. "Challenges in Measuring HPCS Learner Productivity in an Age of Ubiquitous Computing." In Proceedings of the Workshop on Software Engineering and High Performance Computing Applications (held at ICSE 2004). Edinburgh, Scotland.

- Victor Basili, Sima Asgari, Jeff Carver, Lorin Hochstein, Jeffrey K. Hollingsworth, Forrest, Shull, Marv Zelkowitz. A Pilot Study to Evaluate Development Effort for High Performance Computing." University of Maryland Technical Report CS-TR-4588. April 2004.

- Sima Asgari, Victor Basili, Jeffrey Carver, Lorin Hochstein, Jeffrey K. Hollingsworth, Forrest Shull, Marvin Zelkowitz. " High Productivity Computer Systems (HPCS): Empirical studies on Development Time" In the poster session of the 2004 ACM-IEEE International Symposium on Empirical Software Engineering (ISESE 2004) 19-20 August 2004 Redondo Beach CA, USA

Appendix E: Development Time Report from UCSD

This past year Dr. Alan Snavely collaborated with the rest of the HPCS software engineering team and other academics to design and implement a new class curriculum that includes the conducting of HPC software engineering experiments. A sustainable course syllabus was evolved and adopted by several computer science departments. Thereby a body of evidence is being built up, and will continue to be built up ongoing, to answer the question "what is the development time versus performance tradeoff when using today's programming paradigms on HPC platforms?

Dr. Snavely contributed input from the UCSD Computer Science curricula, and presented to the team, then further analyzed, existing class structure and parallel coding assignments in UCSD's graduate level parallel programming courses; thus to help answer the question: "what is current best-practice as to methods taught"? Next the team participated in a pilot study of development time versus execution time for different parallel programming approaches such as MPI, threading (OpenMP), parallel arrays (Co-Array Fortran, UPC) etc. on various HPC platforms. The results of these preliminary investigations, including the class taught by Dr. Mary Hall at USC to which Dr. Snavely provided technical assistance, are being folded into a new course syllabus that will include experiments such as the following to be conducted by Dr. Snavely in the Fall at UCSD (highly simplified example): Assign students a set of parallel programming assignments to code up. Step 1: Code problem(s) in serial. Step 2: Code problem using OpenMP or code problem using MPI. Step 3: Optimize for number of programs solved in a fixed time period (productivity) and/or speed and/or scalability and serial efficiency (performance). Compare the performance and code size and effort of the different approaches. Dr. Snavely provided technical support to Dr. Mary Hall and Dr. Jacqueline Chame in their class at USC taught using SDSC's Blue Horizon Power3 system and incorporated lessons learned into the parallel class (CSE260) that he will teach in the Fall at UCSD.

In Fall 2004 Dr. Snavely will teach the developed syllabus in a parallel class in which students participate in human experiments to quantify productivity by having their programming behaviors monitored as part of course projects to develop HPC problem solutions. He will especially concentrate on quantifying the time-to-correct solution from the programming standpoint vs. additional time required to achieve the best-optimized solution (with respect to wall-clock execution time) tradeoff. The basic approach is a programming contest whereby points are awarded for number or problems solved and/or performance of solution.

Appendix F: Development Time Report from UCSB

The University of California at Santa Barbara accomplished four principle objectives:

1. Developed strategy and materials for an HPC course to be included in the pilot study
2. Conducted a pilot study in an HPC course at UCSB
3. Performed preliminary experiments to extend data collection and analysis techniques
4. Prepared pre-pilot experiments within an HPC environment for use in the studies.

A brief synopsis of each accomplishment follows.

1. Developed strategy and materials for an HPC course to be included in the pilot study

During the Fall of 2003 we redesigned the syllabus of the UCSB graduate course CS 240A, aiming to offer it in Spring 2004. The new course is "an interdisciplinary introduction to applied parallel computing on modern supercomputers. Topics include applications-oriented architectural issues, MPI, OpenMP, parallel Matlab, and parallel numerical algorithms."

The redesign had three goals. First, the new course is to have a distinctly applied and interdisciplinary character. It is to be a required course for first-year graduate students (Master's and PhD) in UCSB's Graduate Specialization in Computational Science and Engineering. In this program, students obtain a MS or PhD in a traditional discipline (Computer Science, Mechanical and Environmental Engineering, Chemical Engineering, Electrical and Computer Engineering, or Mathematics) with a thesis jointly supervised by faculty in different disciplines.

The second goal for the redesigned course was to develop several programming project assignments from CS&E application areas. Implementations were to be done using three different models of computation: message-passing (using MPI); shared-memory (using OpenMP); and novel (using our prototype Matlab*P system).

The goal for the redesigned course was to facilitate quantification of the productivity of the students, for the HPCS pilot study.

2. Conducted a pilot study in an HPC course at UCSB

Working with the UMD team, we conducted a pilot productivity study in CS 240A during the Spring quarter, March to June 2004.

We used the three technologies we had planned, MPI, OpenMP, and Matlab*P. Our two primary computing platforms were a 32-processor Beowulf cluster belonging to UCSB (on which the students ran MPI, 2-processor OpenMP, and Matlab*P) and the Blue Horizon and DataStar HPC systems at the San Diego Supercomputer Center (on which the students ran MPI and multiprocessor OpenMP). The course included 24 first-year graduate students, of which 16 participated in the pilot study. Approximately 2/3 of the

students were from the Computer Science Department and 1/3 from Mechanical and Environmental Engineering.

The students did four individual assignments (as part of the instrumented pilot study). They were:

a. A highly parallel Monte Carlo problem, the Buffon-Laplace needle (in MPI, OpenMP, and Matlab*P).

b. A parallel sorting problem (in MPI and Matlab*P).

c. A discrete simulation problem, the game of Life (in MPI and OpenMP).

d. An irregular scientific kernel, sparse matrix-vector multiplication and conjugate gradient iteration (in a mixture of MPI and Matlab*P).

The students also did term research projects in teams of two or three; the pilot study did not collect productivity data on these projects.

In collaboration with the UMD team, we collected several kinds of data on the students' productivity, including timestamps and source code at each compile, execution timestamps, student-reported effort logs and reasons for recompiles, and pre- and post-study questionnaires.

On the whole, the data collection went well, and we learned a great deal that will serve us during next year's full study. Our ability to collect data on the remote systems at SDSC was limited due to some (unrelated) security issues at the center and due to the decommissioning of Blue Horizon and its mid-study replacement by DataStar.

3. Preliminary experiments to extend data collection and analysis techniques

At the end of the project, we were initiating the analysis of the data we collected. In addition to the analysis being done by the Maryland team, we have started an experiment to determine what can be learned by "replaying the entire programmer experience" -- that is, by using the collected history of program updates and executions to recreate every single test run that was done during development. This is a computationally intensive experiment, but we are after all in the business of using high-performance computing to answer research questions. Our first preliminary experiment is using heuristics to guess the point in the programmer's workflow represented by each compile-and-run cycle. We are planning to pursue this idea further during the coming months in collaboration with the University of Maryland team.

4. Preparation and pre-pilot experiments with an HPC environment for use in the studies.

We are developing a novel HPC programming environment for inclusion in some of the productivity studies. Our environment, called Matlab*P, is a flexible interactive

environment that enables computational scientists and engineers to program high-performance parallel computers. During the reporting period, we have developed support for distributed irregular sparse matrix data and operations in Matlab*P, including a good deal of infrastructure for combinatorial scientific computations. We have also made the preliminary steps in the design of a global address space (GAS) data view that will complement Matlab*P's existing data-parallel SIMD and task-parallel SPMD views and increase the expressiveness of the language.

We used our prototype version of Matlab*P in the course assignments and projects in CS 240A in Spring 2004. Due to its preliminary nature we did not collect productivity data on Matlab*P this time around, but we plan to do so during the next offering of CS 240A in the full study.

PATENTS SUBMITTED IN REPORTING PERIOD:

none

PAPERS SUBMITTED AND PRESENTED IN REPORTING PERIOD:

Ron Choy, Alan Edelman, John R. Gilbert, Viral Shah, and David Cheng. "Star-P: High productivity parallel computing." Accepted to Eighth Annual Workshop on High-Performance Embedded Computing (HPEC-04).

John R. Gilbert and Viral Shah. "Graphs and sparse matrices in an interactive supercomputing environment." Minisymposium talk presented at SIAM Annual Meeting, 2004.

Imran Patel and John R. Gilbert. "Grid*P: Interactive supercomputing on the grid with Matlab." Presented at SIAM Conference on Parallel Processing for Scientific Computing, 2004.

Viral Shah and John R. Gilbert. "Sparse matrices in Matlab*P: Design and implementation." Accepted to 11th Annual International Conference on High Performance Computing (HiPC 2004).

Appendix G: Development Time Report from MIT

The Massachusetts Institute of Technology accomplished four principle objectives:

1. Developed strategy and materials for an HPC course to be included in the pilot study
2. Conducted a pilot study in an HPC course at UCSB
3. Performed preliminary experiments to extend data collection and analysis techniques
4. Prepared pre-pilot experiments within an HPC environment for use in the studies.

A brief synopsis of each accomplishment follows.

1. Developed strategy and materials for an HPC course to be included in the pilot study

During Fall 2003 we modified somewhat the syllabus of the MIT graduate course 18.337/6.338, for Spring 2004. The course is "Applied Parallel Computing" on modern supercomputers. Topics include applications- oriented architectural issues, MPI, OpenMP, parallel Matlab, and parallel numerical algorithms.

We had four goals:

a. The course has a distinctly applied and interdisciplinary character. It is a popular course for first-year graduate students (Master's and PhD) in engineering and science.

b. We developed and improved upon several programming project assignments. Implementations were to be done using three different models of computation: message-passing (using MPI); shared-memory (using OpenMP); and novel (using our prototype Matlab*P system).

c. We designed the course to facilitate quantification of the productivity of the students, for the HPCS pilot study.

d. We put the course on the MIT opencourseware web page.

We kept the Singapore students informed as non-participants

2. Conducted a pilot study in an HPC course at MIT

Working with the UMD team, we conducted a pilot productivity study in 18.337/6.338 during the Spring semester, February to May 2004. We used the three technologies we had planned, MPI, OpenMP, and Matlab*P. Our two primary computing platforms were a 16 processor Beowulf cluster now three years old, and midway through the semester we acquired a second Dell cluster.

The students did three individual assignments (as part of the instrumented pilot study).

They were:

a. A highly parallel Monte Carlo problem, the Buffon-Laplace needle (in MPI, OpenMP, and Matlab*P). **(**Same as Santa Barbara)

b. The grid of resistors problem (in openMP, MPI and Matlab*P).

c. A Laplace Equation Problem (in MATLAB*p, MPI and OpenMP optional).

The students also did term research projects in teams of one, two or three; the pilot study did not collect productivity data on these projects. In collaboration with the UMD team, we collected several kinds of data on the students' productivity, including timestamps and source code at each compile, execution timestamps, student-reported effort logs and reasons for recompiles, and pre- and post-study questionnaires.

On the whole, the data collection went well, and we learned a great deal that will serve us during next year's full study. We had some problems with the move mid semester from Technology Square to the Stata Center. Machines that had worked did not survive the move so well. We had further breakdowns as well. This was statistically unusual when compared with previous years.

3. Performed preliminary experiments to extend data collection and analysis techniques

At the end of the project, the analysis of the data we collected is only beginning. The analysis is being led by the University of Maryland team.

4. Prepared pre-pilot experiments with an HPC environment for use in the studies.

We are developing a novel HPC programming environment for inclusion in some of the productivity studies. Our environment, called Matlab*P, is a flexible interactive environment that enables computational scientists and engineers to program high-performance parallel computers. MATLAB*p is likely to go commercial very soon in the form of Interactive Supercomputing. This will allow greater support possibilities and an industrial strength environment.

We used our prototype version of Matlab*P in the course assignments and projects in 18.337/6.338 in Spring 2004. Due to its preliminary nature we did not collect productivity data on Matlab*P this time around, but we plan to do so during the next offering in the full study.

PATENTS SUBMITTED IN REPORTING PERIOD: none

PAPERS SUBMITTED AND PRESENTED IN REPORTING PERIOD:

Ron Choy, Alan Edelman, John R. Gilbert, Viral Shah, and David Cheng. "Star-P: High productivity parallel computing." Accepted to Eighth Annual Workshop on High-Performance Embedded Computing (HPEC-04).

Parallel Matlab, doing It Right, Alan Edelman, Ron Choy. To be published. IEEE Proceedings.

Appendix H: Report from NGC

A Characterization of a Representative HPCS Benchmark for Intelligence Processing

Final Report

Prepared for Bob Lucas
University of Southern California
Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

under USC Subcontract PO#080930

Robert G. Babb II
Angela Betker
Mark Coffey
Bruce Lenell
John Szaro

Northrop Grumman Mission Systems
17455 E Exposition Dr
MS AUC1/41
Aurora, CO 80017

Robert.Babb@ngc.com
Angela.Betker@ngc.com
Mark.Coffey@ngc.com
Bruce.Lenell@ngc.com
John.Szaro@ngc.com

21 June 2003

1. INTRODUCTION

The "p" in the DARPA HPCS program [http://www.darpa.mil/ipto/research/hpcs/] acronym stands not for "performance" but "productivity" ("High Productivity Computing Systems"). High productivity is considered to result from a combination of:

- **Performance:** Improve the computational efficiency and performance of critical national security applications
- **Programmability:** Reduce cost and time of developing HPCS application solutions
- **Portability:** Insulate research and operational HPCS application software from system specifics
- **Robustness:** Deliver improved reliability to HPCS users and reduce risk of malicious activities

This study was undertaken as a first step in understanding the characteristics of computationally intensive algorithms in one area of intelligence processing. Since this is meant to be an unclassified report, of necessity most aspects of the domain can be discussed only in somewhat general terms, and the specific algorithm chosen for study is described mathematically, without discussion of its purpose, or where it fits into the larger intelligence processing environment. The algorithm does represent a key link in operational intelligence systems of critical national importance.

2. HIGH PERFORMANCE ALGORITHM DEVELOPMENT FOR THIS DOMAIN

In general terms, the following are some key characteristics of the way algorithms are developed and implemented within one subset area of intelligence processing.

Programming Language: C or C++, although some areas are experimenting with Java, especially for J2EE-style network programming and user interface support. There is also some legacy code still running that was written in Fortran, Assembler, and other languages.

Lifetime of a single application code: up to 20 years

Development Teams: usually between 2 and 15 people for any single algorithm or application area.

Development Environment: The systems that these algorithms are part of are themselves very large and distributed across dozens or hundreds of compute and data servers. The end users of the systems are both local to the main processing chains, and widely distributed geographically. Very high data rates must be supported for input, and in some instances on output as well. The requirements (and changing requirements) for these systems thus tend to stress whatever compute, data, and network capabilities exist at any particular point in time, and this has a large impact on the way applications are

developed, both for modifications to existing systems, and introduction of new types of processing.

Development Approach: This can best be described as bursts of concentrated efforts dealing with the "crisis of the moment". Continually "shoe-horning" applications onto platforms tends to lead to complex, expensive to maintain code, and tends to work against the HPCS "programmability" objective mentioned above. Because these systems are so large, and also due to security considerations, individual programmers rarely have a good understanding of the "big picture" that their piece fits into. This has two implications:

(1) Programmers tend to fall into "sub-optimization" traps while attempting to make their part of the processing "ultra efficient" (at the expense of code simplicity, clarity, and maintainability), and sometimes with negative impacts on the overall processing effectiveness.

(2) Since there tends to be poor coordination, understanding and control of the overall system architecture, system processing bottlenecks and imbalances can develop that must be "worked around" with a great deal of (heroic) programming effort, which tends to further complicate system and application program development efforts.

There is also a larger organizational effect on the way these systems have evolved, since various parts of the processing are developed at various times by competing contractors and sometimes for competing government organizations. This can have a further "sub-optimizing" effect on the code bases on a larger scale than the individual programmer sub-optimization mentioned above.

3. A REPRESENTATIVE (ABSTRACT) ALGORITHM

The algorithm described in this section has several interesting general characteristics. First, the amount of computing required by the problem grows exponentially in the size of the sets of elements processed. This means that an exhaustive computing approach is not possible except in trivially small cases, and variants of the algorithm compete on how well the heuristics they employ perform. Since the actual "answer" is in general not known, it is not even possible to determine how well any of the heuristics perform compared to "the answer" for a particular input data set.

The problem is a constrained set partitioning problem where the "hardest" constraint test (Step 4 below) is only performed when the easier constraints have not been conclusive, because it is computationally much more expensive (matrix operations on floating point values) than the easier constraints. It is also the most powerful constraint in that it can result in an answer that can be output immediately for further processing. This problem is also a "soft" real-time problem in that there is a time value attached to any answers that can be determined within, for example, 15 minutes, and any answers found after that time are of rapidly decreasing value, and so are generally not computed. The current form of the algorithm has been driven largely by this real-time constraint. More comprehensive set partitioning approaches to the problem are also of considerable interest.

Algorithm 1: Set Partitioning Under Constraints Including Nonlinear Least Squares

Let S be a totally ordered set of size 1,000 to 50,000 elements. The elements of S are processed sequentially and individually. The algorithm terminates when all of the elements of S have been processed.

The algorithm uses an undirected graph G whose vertices are subsets of S. If A and B are subsets of S and vertices of G, then there is an edge connecting vertices A and B in G if and only if $A \cap B \neq \varnothing$ (*Edge Property*). There is also a total ordering on the vertices of G.

Let s[i] be the next element of S to be processed. The algorithm traverses the vertices of G in order, and for each vertex v[j] computes the value of a function f that determines the next step the algorithm will take. The function f has the form

f: $S \times G \rightarrow \{0,1,2\}$
and the algorithm behavior in response to the various possible values returned by f is:

f(s[i],v[j]) = 0 : Move to the next vertex v[j+1] and continue by computing f(s[i],v[j+1]).

= 1 : Duplicate the vertex v[j] creating a new vertex, add element s[i] to the new vertex, add the new vertex immediately before v[j] in the ordering of vertices of G, renumber the vertices of G, recompute the edges of G to maintain the *Edge Property*, and continue with f(s[i],v[j+2]).

= 2 : Add s[i] to v[j], recompute the edges of G to maintain the *Edge Property*, delete the immediate neighbors of v[j], recompute the edges of G to maintain the *Edge Property* after the deletion, and continue by computing f(s[i+1], v[1]) i.e. processing of element s[i] is complete.

If the entire graph G is traversed and no function evaluation of f has returned 2, create a new vertex v[n] in G (last in the ordering of vertices) consisting of s[i].

The behavior of the function f is as follows. f is evaluated in four steps of increasing computational complexity and decreasing probability of being executed.

Step 1: (Relative probability of returning f = 0, vs. proceeding eventually to Step 4: 1,000,000).
Computational Complexity: Small numbers of comparisons of small character strings (<= 10 strings of <= 10 characters).
Result: Either return f = 0 or proceed to Step 2.

Step 2: (Relative probability of returning f = 0, vs. proceeding eventually to Step 4: 10,000).

Computational Complexity: Lookups of thresholds in tables of 100-1,000 entries in memory, scalar differences and comparisons against those thresholds.
Result: Either return f = 0, or proceed to Step 3.

Step 3: (Relative probability of returning f = 0, vs. proceeding to Step 4: 100)
Computational Complexity: Matrix sums and multiplications of small matrices (order ≤ 10),
Result: Either return f = 0, or proceed to Step 4.

Step 4: (Relative probability of being executed: 1).
Computational Complexity: Nonlinear least squares for ~30 iterations on matrices of order ≤ 10. Multiple matrix operations of order ≤ 30. Lookups against significantly larger tables in memory per iteration, compared to the tables in Step 2. All matrices and arithmetic operations are on double precision floating point values.
Result: Return f = 0, 1, or 2.

4. FURTHER WORK

The algorithm described above represents only one of several critical computationally intensive algorithms in this sub-area of intelligence processing. Consideration should be given to developing a similar unclassified characterization and concrete benchmarks for those algorithms as well.